DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS
55

ATTEMPTO CONTROLLED ENGLISH AS A SEMANTIC WEB LANGUAGE

KAAREL KALJURAND

TARTU 2007

Faculty of Mathematics and Computer Science, University of Tartu, Estonia

Dissertation is accepted for the commencement of the degree of Doctor of Philosophy (PhD) on November 26, 2007 by the Council of the Institute of Computer Science, Faculty of Mathematics and Computer Science, University of Tartu.

Supervisor:

PhD Norbert E. Fuchs
University of Zurich
Zurich, Switzerland

Co-supervisor:

PhD Kaili Müürisep
University of Tartu
Tartu, Estonia

Opponents:

PhD Uta Schwertel
Ludwig-Maximilians-University of Munich
Munich, Germany

PhD, professor Tanel Tammet
Tallinn University of Technology
Tallinn, Estonia

Commencement will take place on January 10, 2008.

ISSN 1024–4212
ISBN 978-9949-11-781-9 (trükis)
ISBN 978-9949-11-782-6 (PDF)

Tartu Ülikooli Kirjastus
www.tyk.ee
Tellimus nr 525

Contents

Chapter 1

Introduction

The Semantic Web is an enhancement of the World Wide Web in which web content is expressed in a form that can be understood and used by software agents, permitting them to find, share and integrate information more easily. Research in the area of the Semantic Web has created a plethora of formal languages for representing knowledge in the form of ontologies, business rules, search queries, etc. While these languages comply with the basic design decisions of the Semantic Web architecture (e.g. they have a formal semantics, their syntax is based on the Extensible Markup Language (XML), they use Uniform Resource Identifiers (URI/IRI) for naming), the human aspects of these languages (learnability, readability, writability) have received little attention. At the same time it is expected that these languages will be widely used in the future, not only by machines but also humans. In this thesis, we argue that this wide adoption can be made possible only by bringing the various formats closer to the end user, somebody who has usually no training in formal methods.

In this thesis, we advocate the idea of using controlled English as a Semantic Web language. On the one hand, such English allows the user to create Semantic Web content in a user-friendly, yet logically precise way. On the other hand, it provides an ambiguity-free language for explaining (verbalizing) the existing Semantic Web content.

For the controlled natural language to be used as a Semantic Web language, we have chosen Attempto Controlled English (ACE). We will show how ACE overcomes the problems of usability that the existing Semantic Web languages suffer from, while preserving most of the good properties that these languages offer.

The existing Semantic Web languages under focus are fragments of first-order logic (FOL) with the classical first-order logic semantics, namely Web Ontology Language (OWL) and Semantic Web Rule Language (SWRL).

In the main part of the thesis, we show that a subset of ACE is as expressive as OWL, but — as a fragment of English — offers better usability for domain experts with no training in logic. We then discuss how our approach can be extended to cover other Semantic Web languages — rule languages and query languages — with no major change in the syntax i.e. the extended language will remain a natural subset of ACE.

1.1 Semantic Web languages

Figure 1.1 shows the latest thinking about the Semantic Web stack where each language is represented by a block that is supported by other blocks of less expressive languages. At the basis of the stack are located completed standards — "URI/IRI" that provide means for global identification of objects (e.g. web-pages), "XML" as the prominent serialization syntax of the Semantic Web languages, "RDF" for simple triple-based data modeling. The higher levels of the stack (e.g. abstract frameworks such as "Proof" and "Trust") have not been fully and concretely worked out yet.

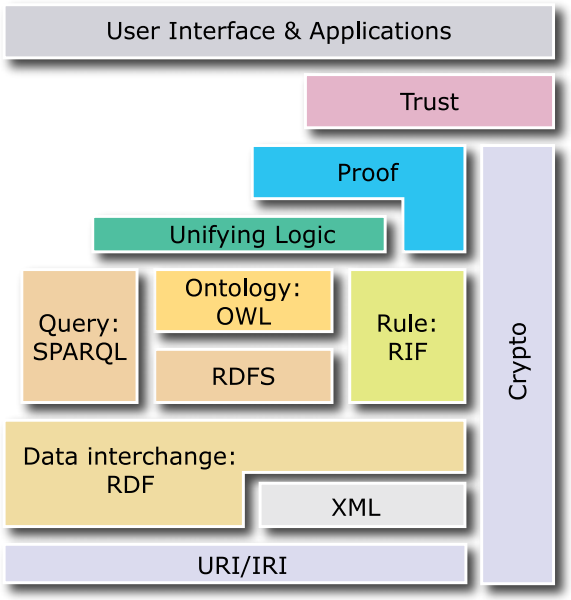


Figure 1.1: The latest (March 2007) version of the Semantic Web stack as envisioned by the World Wide Web Consortium (`http://w3.org`). The image originates from `http://www.w3.org/2007/03/layerCake.svg`.

Our interest in this stack is in the ontology languages (OWL and RDFS), rule languages and interchange formats (RIF, SWRL, ...), and query languages (SPARQL, ...), which are conceptually side by side. Special focus on this diagram is given to the user interfaces which matches our approach of using ACE as a Semantic Web language — as ACE does not introduce any new logic to the Semantic Web stack, it really targets the user interface block.

1.2 Why ACE?

ACE is a controlled English that has been under development since 1995 in the Attempto project at the University of Zurich, and has had many users and use-cases over the years. It is an expressive language, both syntactically and semantically, and has a formal logi-

cal basis, i.e. it offers natural language counterparts to common logical operators ('and', 'or', 'not', 'if ... then ... '), variables (i.e. anaphoric references), quantification (e.g. determiners 'every' and 'no', various plural constructions), etc. Many of these constructs can be expressed in syntactically different ways, e.g. negation is not limited to sentences but can be used also with relative clauses, verb phrases, and nouns. Anaphoric references come in the form of pronouns and definite noun phrases, and relative clauses offer even more ways of binding.

Since 2004, the author of this thesis has been working in the project REWERSE[1] (Reasoning on the Web with Rules and Semantics), where his task has been to help to extend and modify ACE for the purposes of the Semantic Web. As a result of this work, ACE has had several version updates. In this thesis, the most recent version of ACE — version 6.0 — is going to be used.

1.3 Why OWL?

OWL is a knowledge representation language that is largely based on expressive description logics (see [BCM+03]). OWL is a mature standard (although a minor update to the 2004 standard, OWL 1.1, is expected in the year 2009[2]) that is widely used and supported with many ontology editors, reasoners, user communities, and use-cases. As OWL is an expressive language, its syntax and semantics is hard for the potential users to learn and use. The various OWL syntaxes and front-ends have so far not solved this usability issue. At the same time, OWL is quite close to natural language (e.g. OWL class descriptions can be seen as English noun phrases, a similar observation is made for an OWL-like language CLASSIC in [BMPS+91]) and thus suggests a natural language based syntax.

1.4 Contributions

In this thesis, we provide a fresh look to ontology languages and ontology engineering by discussing how OWL ontologies can be understood in natural language, and which OWL constructs are important from the natural language point of view.

We describe a link between a formal ontology language (OWL) and a controlled natural language (ACE). As a result, OWL will profit from ACE as OWL currently has a steep learning curve — many people who otherwise like the idea of the Semantic Web, fear OWL and do not know how to use it. Verbalizing OWL ontologies in natural language and presenting the result as plain text has several advantages. Presentation in English is beneficial in various ways — by hiding the formal syntax of OWL, the ontologies become understandable by any speaker of English. Using controlled English as *lingua franca* can improve the communication between knowledge engineers and domain experts who both work on the same ontology. A verbalization language can be

[1] http://rewerse.net

[2] http://www.w3.org/2007/06/OWLCharter.html

useful also to OWL experts who have no problems with formal syntaxes, because now they can "think aloud" about the formulas. Also, it becomes possible to apply existing natural language processing tools such as spell checking and speech synthesis to formal ontologies, and thus to explore new ways of quality control of and access to ontologies. There will be less need for dedicated and possibly complex ontology editors because plain text can be viewed and modified in any text editor. Plain text can also be easily stored, compared and searched with existing general tools.

The link between ACE and OWL is also beneficial for ACE, as ACE will gain an alternative formalization, visualization and consistency checking (and other support that automatic OWL reasoners provide). A subset of ACE will be proved to be decidable (as OWL 1.1 that we target is a decidable language). Also, OWL reasoners will become ACE reasoners, OWL editors will become ACE editors, the OWL community could become an ACE community. Existing OWL ontologies will become ACE texts, i.e. ACE systems will be able to use OWL knowledge bases as background knowledge.

A bidirectional mapping between a formal language and a controlled natural language is also useful for teaching purposes, either for teaching OWL or for teaching ACE.

Although the ontology frameworks and the rule frameworks of the Semantic Web overlap regarding their concepts and their expressivity [GHVD03], the current syntax for ontologies (i.e. OWL) is radically different from the syntax for rules (e.g. SWRL, RuleML). The Semantic Web query languages use yet another syntax. In contrast, ACE can express ontologies, rules and queries in one and the same syntax.

1.5 Organization of the thesis

This thesis has the following parts.

- In chapter 2, we introduce the idea behind controlled natural languages and give an overview of Attempto Controlled English (ACE);
- in chapter 3, we introduce the OWL language, its syntax and semantics, OWL editors, ways of authoring OWL ontologies, and the shortcomings of current approaches;
- in chapter 4, we review the work related to using controlled English for the Semantic Web;
- in chapter 5, the main part of this thesis, we describe our solution to expressing OWL in ACE in a bidirectional way;
- in chapter 6, we discuss some extensions to our approach, which enable us to express OWL data properties, query languages, and the rule language SWRL;
- in chapter 7, we describe an implementation of the translation between ACE and OWL;

- in chapter 8, we evaluate the designed mapping between ACE and OWL with a case study of various real-world ontologies;
- in chapter 9, we draw conclusions and describe future work.

The main results of this thesis have been published in [FKS05, FKK+06b, KF06a, KF06b, FKK06a, FKK07d, KF07, FKK07e]. The implemented software and its documentation is available from the Attempto project website[3].

[3] http://attempto.ifi.uzh.ch

Chapter 2

Controlled natural languages and Attempto Controlled English

2.1 Introduction

Controlled natural languages (CNLs) are subsets of natural languages, obtained by restricting the grammar and vocabulary in order to reduce or eliminate ambiguity and complexity. Traditionally, controlled natural languages fall into two major types: those that improve readability for human readers (e.g. non-native speakers), and those that enable reliable automatic semantic analysis of the language.

The first type of languages (often called "simplified" or "technical" languages), for example ASD Simplified Technical English[1], Caterpillar Technical English, IBM's Easy English, are used in the industry to increase the quality of technical documentation, and possibly simplify the (semi-)automatic translation of the documentation. These languages restrict the writer by general rules such as "write short and grammatically simple sentences", "use nouns instead of pronouns", "use determiners" and "use active instead of passive", and often use a predefined vocabulary without synonyms.

The second type of languages have a formal logical basis, i.e. they have a formal syntax and semantics, and can be unambiguously mapped to an existing formal language, such as first-order logic. Thus, those languages can be used as knowledge representation languages, and writing of those languages can be supported by fully automatic consistency and redundancy checks, query answering, etc. Parsing the second type of languages is usually strictly rule-based, and furthermore, the user is informed about the rules and what they accept and generate. Therefore, although the user cannot change the rules, he/she has full control over the output of the parser, as he/she can reformulate the input. In this sense, the approach is different from the approach taken in wide-coverage natural language parsers (e.g. the Collins parser) which deliver the output for any input, and try to "guess" what output the user expects (based on statistical information about resolving various ambiguities).

[1] http://www.simplifiedenglish-aecma.org/Simplified_English.htm

In this thesis, we are only interested in the latter type of controlled natural languages. In the following sections we introduce Attempto Controlled English (section 2.2); its construction rules (section 2.3); its interpretation rules (section 2.4); the DRS language for the meaning representation of ACE texts (section 2.5); and finally review work on other controlled natural languages (section 2.7).

2.2 Attempto Controlled English

Attempto Controlled English (ACE) is a subset of English that can be converted into Discourse Representation Structures (DRS) — a syntactic variant of first-order logic — and automatically reasoned about (see [FKS06] for a general overview). ACE offers language constructs like

- singular and plural countable nouns (‘a man’, ‘some men’);
- mass nouns (‘some water’);
- existential and universal quantification (‘there is a man’, ‘every man’);
- numerical quantifiers (‘at least 5 men’, ‘less than 5 men’);
- indefinite pronouns (‘everybody’, ‘somebody’);
- relative clauses (‘a man that owns a car’);
- active and passive verbs (‘a man owns a car’, ‘a car is owned by a man’);
- various forms of anaphoric references to noun phrases (‘he’, ‘himself’, ‘the man’, ‘X’);
- negation, conjunction and disjunction of noun phrases, verb phrases, relative clauses and sentences.

ACE is defined by a small set of construction rules that define its syntax and a small set of interpretation rules that disambiguate constructs that in full English might appear ambiguous. Word meanings in ACE texts are underspecified, the user is expected to describe the meaning of words by ACE sentences or import them from an existing formal ontology.

ACE can be used in a number of knowledge engineering scenarios such as building a knowledge base, querying an existing knowledge base, etc., where traditional formal languages are otherwise needed. The intention behind ACE is to provide domain specialists with an expressive knowledge representation language that is easy to learn, use and understand.

Using ACE is supported by various tools, most importantly a parser[2] that translates ACE texts into DRSs [Höf04, FKK07c]. Additional tools include a reasoner [FS03],

[2]The ACE parser developed in the Attempto project, ACE Parsing Engine (APE), can be used via a web-client `http://attempto.ifi.uzh.ch/ape/`, or programmatically as a web-service described at `http://attempto.ifi.uzh.ch/site/docs/ape_webservice.html`

a paraphraser [FKS05, FKK06a], and a predictive editor (used as part of AceWiki [FKK07e]). Although ACE can be written and read without tool support, various tools help in assuring that the resulting ACE texts is what was intended. (Note that an ACE parser is needed in any case, to be able to claim that a text is really in ACE.) Experience gained from teaching ACE to university students shows that one can learn to write ACE in a few days. On the other hand, being based on English, ACE can be immediately read by anybody familiar with English.

2.3 ACE construction rules

The ACE construction rules [FKK07a] define admissible sentence structures for ACE. The following sections describe words allowed in ACE (section 2.3.1), phrases composed of such words (section 2.3.2), declarative and interrogative sentences composed of such phrases (sections 2.3.3, 2.3.4), and ACE texts and queries composed of such sentences (section 2.3.5).

In the following, we are not describing the construction rules of full ACE but only the rules that define the subset of ACE that is relevant for this thesis. For example, the support that ACE provides for adjectives and adverbs is not discussed here in detail, but only pointed out in section 2.3.6. For the full description of ACE construction rules, see [FKK07a].

2.3.1 Words

ACE function words and some fixed phrases are predefined and cannot be changed by users. Predefined function words are determiners, quantifiers, prepositions, coordinators (‘and’, ‘or’, ‘, and’, ‘, or’), negation words (‘no’, ‘not’, ‘does not’, ‘is not’, ‘do not’, ‘are not’), pronouns, query words, copula *be*, Saxon genitive marker *’s*, and natural numbers (‘1’, ‘2’, . . . ; ‘one’, ‘two’, . . . , ‘twelve’). Predefined fixed phrases are *there is/are . . . such that*, and *it is false that . . . .*

ACE content words are nouns, proper names, and verbs. Content words can be simple (‘code’), or compound with hyphen (‘zip-code’).

2.3.2 Phrases

ACE supports noun phrases and verb phrases.

Noun phrases

ACE noun phrases are

- singular countable noun phrases: *a card, the card, 1 card, one card, no card, every card, each card, not every card, not each card*;
- plural countable noun phrases: *the cards, some cards, all cards, no cards, nothing but cards, 3 cards, three cards, at least 3 cards, at most 3 cards, more than 3 cards,*

less than 3 cards, exactly 3 cards, not 3 cards, not three cards, not at least 3 cards, not at most 3 cards, not more than 3 cards, not less than 3 cards, not exactly 3 cards;

- proper names: *John, Mr-Miller, Pi*;
- numbers and strings: *12, -2, 3.141, "", "this is a string!"*;
- non-reflexive pronouns (in nominative and accusative case): *it, he, she, he/she, they, it, him, her, him/her, them*;
- reflexive pronouns: *itself, himself, herself, himself/herself, themselves*;
- indefinite pronouns: *someone, somebody, something, no one, nobody, nothing, everyone, everybody, everything, not every one, . . .* ;
- variables: *X, Y1*.

It is important to note that common nouns (e.g. 'card') always come with a determiner ('a', 'the', 'every', 'no', 'at least 2', . . .).

of-prepositional phrases ('of' followed by a noun phrase), Saxon genitives ('John's'), non-reflexive possessive pronouns (*its, his, her, his/her, their*), and reflexive possessive pronouns (*its own, his own, her own, his/her own, their own*) can optionally be attached to the noun, to construct a more complex noun phrase.

(2.1) a card of John

(2.2) John's card

(2.3) his own card

Variables can optionally occur in the apposition of a countable noun phrase, or an indefinite pronoun, to construct a more complex noun phrase.

(2.4) a card X

(2.5) at least 3 cards X

(2.6) something X

A powerful way to construct more complex noun phrases is provided by relative clauses. A relative clause can optionally follow a noun, a proper name, a pronoun, or a variable. Both subject- and object-extracted relative clauses are supported.

(2.7) a customer who enters a card

(2.8) John who knows Mary

(2.9) it that borders Estonia

(2.10) everything that a brain-cell contains

(2.11) X which is a card

(2.12) a man a car of who is liked by Mary

(2.13) a man whose car is liked by Mary

Relative clauses can be coordinated (i.e. used in either conjunction or disjunction).

(2.14) a customer who enters a card and who Mary knows or a friend of who is a manager

A relative clause can also be negated.

(2.15) a customer who does not enter a card and who Mary does not know

A noun phrase is anaphoric if it starts with the definite determiner 'the', or starts with a possessive pronoun, or is headed by a pronoun, variable, or a proper name. Anaphoric noun phrases are interpreted as making a reference to an earlier noun phrase in the text (see section 2.4.3).

Verb phrases

ACE verbs are either intransitive ('wait'), transitive ('enter *something*'), or ditransitive ('give *something* to *somebody*' or 'give *someone something*'). For the subset that we a considering here, we only allow transitive verbs.

ACE verbs are in 3rd person singular or plural, in indicative mood, and in simple present tense. Both active and passive verbs can be used but passive constructions must include a prepositional phrase '... by ...'. Phrasal particles and prepositions that introduce a complement of a transitive verb, must be hyphenated to the verb.

(2.16) John enters a card.

(2.17) A card is entered by John.

(2.18) John fills-in a form.

(2.19) A form is filled-in by John.

The copula verb 'be' is similar to transitive verbs as it takes a complement.

(2.20) John is a customer.

(2.21) John's age is 32.

A pair of verb phrases can be coordinated (i.e. used in either conjunction or disjunction).

(2.22) a customer enters a card and knows Mary or is a manager

A verb phrase can also be negated.

(2.23) John does not enter a card.

(2.24) John is not a manager.

2.3.3 Declarative sentences

Simple declarative sentences are a concatenation of a noun phrase with a verb phrase, and end with a full stop.

(2.25) A customer enters a card.

(2.26) Every customer enters a card.

In addition to this general sentence structure, it is possible to create well-formed sentences with a single noun phrase, prefixed by the fixed phrase *there is/are*.

(2.27) There is a customer that enters a card.

(2.28) There are more than 6 customers.

Composite declarative sentences are recursively built from simpler sentences (without the full stop) using the predefined constructors: coordination, negation, global quantification, *if-then* subordination. Composite sentences end with a full stop.

Sentence coordination allows to combine sentences into conjunctions and/or disjunctions.

(2.29) There is a man X, and every woman likes the man X or every woman hates the man X.

Sentence negation ('it is false that') can be used in sentence initial position to negate the complete sentence.

(2.30) It is false that John likes Mary.

(2.31) It is false that John is a manager and that John likes Mary.

Global existential quantification (*there is/are ... such that*) and global universal quantification (*for every ...*) can be used in sentence initial position to quantify over the complete sentence.

(2.32) There is a code such that every clerk enters it.

(2.33) For every code a clerk enters it.

(2.34) For every code there is a clerk such that he enters it.

Conditional sentences (*if-then* sentences) are built with the help of function words 'if' and 'then' where both must be followed by a sentence.

(2.35) If John enters a card then the clerk accepts it.

2.3.4 Interrogative sentences

ACE allows two simple forms of interrogative sentences: *yes/no* queries (that start with 'does', 'do' or 'is', 'are') and *wh*-queries (that contain 'who', 'what'). Every interrogative sentence ends with a question mark.

(2.36) Does John enter a card?

(2.37) Is John a manager?

(2.38) Who enters what?

(2.39) A friend of who is a manager?

In those sentences, 'who' and 'what' occupy the noun phrase position. 'Does' and 'Is' turn a declarative sentence into a boolean interrogative sentence.

2.3.5 ACE texts and queries

Every ACE text is a sequence of simple or composite declarative sentences. For example

(2.40) There is a customer that owns a car. It is false that John likes the car.

Every ACE query consists of a sequence of simple or composite declarative sentences, followed by exactly one interrogative sentence.

(2.41) There is a customer that owns a car. Who owns what?

2.3.6 More features

There are several features of ACE which are less important in the context of this thesis. We briefly list those features here and refer the reader to [FKK07a] for a detailed discussion.

- In addition to countable noun phrases, ACE supports mass noun phrases, e.g. 'some water', 'all sand'; and measurement noun phrases, e.g. '3 liters of water'.
- Conjunction of noun phrases is allowed and interpreted as creating a "plural object", e.g. 'John and Mary'.
- Plural noun phrases can be prefixed by 'each of', e.g. 'each of 3 cards' to support expressing the distributive reading of the noun phrase.
- Noun phrases can be pre-modified by adjectives and adjective coordinations ('a rich and happy man'), where adjectives can be positive, comparative, or superlative. Adjectives are also allowed as copula complements ('a man is rich', 'John is richer than Bill').

- Verb phrases can be modified by adverbs and prepositional phrases. They follow the verb and its complements (if present). Adverbs can also precede the verb. Query words 'how', 'where', and 'when' ask for modifiers of the verb in interrogative sentences.
- Sentences and verb phrases can be used in modal (possibility and necessity) constructions, e.g. 'it is possible that John likes Mary', 'John can like Mary'.
- Sentences can be used in *that*-subordination, e.g. 'John thinks that Mary likes him.'.
- A special fixed phrase 'it is not provable that' has been introduced to support "negation as failure" that is known from many knowledge representation languages.
- Imperative sentences ending with an exclamation mark can be used to issue commands, e.g. 'John, enter the card!'.
- Full ACE also incorporates a simple macro facility — one can label a sentence and use the defined label to refer to the sentence later in the text.

2.4 ACE interpretation rules

Each ACE sentence has only one meaning. However, if interpreted in full English, a sentence which is syntactically in ACE can be seen as having many meanings. The purpose of ACE interpretation rules is to clarify which of the conceivable meanings a sentence has in ACE. The full list of rules is given in [FKK07b].

In general, the expectation is that the interpretation of ACE constructs is clear to English speakers and that the user-level documentation of ACE semantics does not have to describe how the sentences are mapped into their logical form that provides the explicit formal semantics. Still, there are several aspects of ACE sentences (mostly related to quantification, coordination and anaphoric references) that need a user-level discussion.

2.4.1 Quantifiers and their scope

ACE universal quantifiers are 'every' (alternatively 'each', 'all') and 'no', and the existential quantifiers are 'a' (alternatively 'an', 'some') and the numerical quantifiers (e.g. 'at least 2'). These quantifiers are used as noun determiners. In addition, there are sentence initial quantifiers 'for every' ('for each', 'for all') for universal quantification, and 'there is/are ... such that' for existential quantification. Ambiguity of the quantification words is not allowed. This means that e.g. 'a' cannot be used as a universal quantifier even though in English it is sometimes used in this way.

The textual position of a quantifier opens its scope that extends to the end of the sentence; in sentence coordinations the scope extends to the end of the respective coordinated sentence. This means that the classical scope ambiguity example

(2.42) Every customer types a code.

is interpreted in ACE as the existential quantifier ‘a’ being within the scope of the universal quantifier ‘every’. To express the other scoping, one would have to say

(2.43) There is a code such that every customer types the code.

2.4.2 Coordination

Similarly to many logics, ACE provides coordinators ‘and’ and ‘or’, both taking two arguments. The resulting binding order ambiguity is controlled as in logics where ‘and’ binds stronger than ‘or’. This, however, can be reversed by prefixing ‘and’ with a comma (i.e. by using the so-called “comma and”). An even weaker binder is “comma or”. Thus we have the following binding order (where $a \succ b$ stands for “a binds stronger than b”):

(2.44) $$and \succ or \succ ,and \succ ,or$$

In the following example, the scope of the disjunction is denoted by brackets.

(2.45) A client {enters a UBS-card or enters a ZKB-card}, and types a code.

Using coordination in relative clauses, results in an attachment ambiguity in English. For example, in the sentence

(2.46) Every man owns a dog that likes a cat that likes a mouse and that eats a bone.

the last relative clause ‘that eats a bone’ can syntactically attach to both ‘dog’ and ‘cat’. I.e. in full English, this sentence is considered ambiguous between the readings

(2.47) Every man owns a dog_d $that_d$ likes a cat_c $that_c$ likes a mouse and $that_d$ eats a bone.

(2.48) Every man owns a dog_d $that_d$ likes a cat_c $that_c$ likes a mouse and $that_c$ eats a bone.

(where co-indexing explicitly shows the relative clause attachment) and the attachment is determined probably by the word sense properties of the word ‘bone’ which selects the relative clause to attach to ‘dog’ (because dogs, and not cats eat bones) (reading 2.47). ACE, on the other hand, interprets all relative clauses to attach to the most recent noun (reading 2.48).

2.4.3 Anaphora resolution

The rules of anaphora resolution specify how anaphoric noun phrases reference earlier noun phrases in the text. Proper names like ‘John’ or ‘Mr-Miller’ always denote the same object and thus serve as their own anaphoric references; in the other cases (pronouns, definite noun phrases and variables), resolution of anaphoric references is governed by accessibility, recency, specificity, and reflexivity.

Accessibility

A noun phrase is not accessible if it occurs in a negated sentence.

(2.49) John does not enter a card. *A clerk takes the card.

A noun phrase is not accessible if it occurs in a universally quantified or *if-then*-sentence. However, a noun phrase in the *if*-part of a conditional sentence is accessible in the *then*-part.

(2.50) Every customer has a card. *A clerk takes the card.

(2.51) If there is a customer then he has a card. *A clerk takes the card.

(2.52) If a customer has a card then a clerk takes the card.

A noun phrase in a disjunction is only accessible in subsequent disjuncts.

(2.53) A customer enters a card or drops the card. *A clerk takes the card.

Recency and reflexivity

If the anaphor is a non-reflexive personal pronoun ('he', 'him', ...), or a non-reflexive possessive pronoun ('his', ...), then the anaphor is resolved with the most recent accessible noun phrase that agrees in gender and number, and that is not the subject or an object of the sentence in which the anaphor occurs.

(2.54) $John_j$ has a $card_c$. Bob sees him_j and takes it_c.

(2.55) *John sees his wife.

If the anaphor is a reflexive personal pronoun ('herself', ...), or a reflexive possessive pronoun ('her own', ...), then the anaphor is resolved with the subject of the sentence in which the anaphor occurs if the subject agrees in gender and number with the anaphor.

(2.56) $Mary_m$ takes $her\ own_m$ mirror and looks-at $herself_m$.

Specificity

If the anaphor is a definite noun phrase then it is resolved with the most recent and most specific accessible noun phrase that agrees in gender and number.

(2.57) There is $a\ ball\ of\ a\ boy_{bb}$. There is $a\ ball\ of\ a\ girl_{gb}$. John sees $the\ ball_{gb}$. Mary sees $the\ ball\ of\ a\ boy_{bb}$.

(2.58) There is $a\ man\ that\ is\ a\ manager_{mm}$. There is $a\ man\ that\ is\ a\ programmer_{mp}$. John sees $the\ man_{mp}$. Mary sees $the\ man\ that\ is\ a\ manager_{mm}$.

If the anaphor is a variable then it is resolved with an accessible noun phrase that has the variable as apposition.

(2.59) John has *a card* X_x and has *a card* Y_y. Mary takes *the card*$_y$. Bob takes *the card* X_x. Harry takes Y_y.

2.5 Discourse Representation Structures

2.5.1 Introduction

Discourse Representation Theory ([Kam81, KR93, BB99]) describes how a set of anaphorically interlinked natural language sentences can be represented and systematically translated to Discourse Representation Structures (DRSs), a variant of first-order logic. The main advantage of Discourse Representation Theory over previous approaches to natural language semantics lies in the proper treatment of anaphoric references (e.g. pronouns) and discourse (anaphorically interlinked sentences). In the following, we look at the syntax and semantics of DRSs and how ACE sentences are represented by them.

2.5.2 Syntax

The syntax of DRSs and DRS conditions is defined by the following rules.

- If $x_1,\ldots,x_n$ are discourse referents and $C_1,\ldots,C_m$ are DRS conditions, then the following is a DRS: $\boxed{\begin{array}{|c|}\hline x_1 \ldots x_n \\ \hline C_1 \ldots C_m \\ \hline \end{array}}$.
- If R is a relation symbol of arity n, and $x_1,\ldots,x_n$ are discourse referents or constants, then $R(x_1,\ldots,x_n)$ is a DRS (atomic) condition.
- If B_1 and B_2 are DRSs, then $B_1 \Rightarrow B_2$ and $B_1 \vee B_2$ are DRS conditions.
- If B is a DRS, then $\neg B$ is a DRS condition.
- Nothing else is a DRS or DRS condition.

In the following, we sometimes refer to DRSs as "DRS boxes". The DRS conditions $B_1 \Rightarrow B_2$, $B_1 \vee B_2$, and $\neg B$, are called implication-condition, disjunction-condition, and negation-condition, respectively. Their arguments (B, B_1, B_2) are called sub-DRSs or embedded DRSs. The arguments of an implication condition are called an *if*-box and a *then*-box. The single DRS which is not embedded into any condition is called a top-level DRS, and its discourse referents are top-level discourse referents.

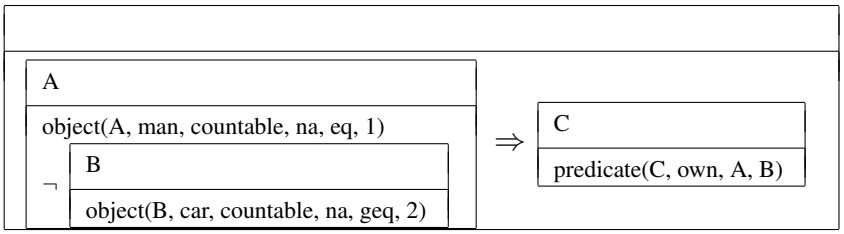
An important concept related to DRSs is accessibility. Accessibility describes restrictions on how conditions can use discourse referents as arguments. Only those discourse referents that are declared in an accessible DRS box can be used as arguments. Let B_1

and B_2 be DRSs. Conditions in the DRS B_2 can access the discourse referents declared in B_1 if and only if

- B_1 and B_2 denote the same DRS; or
- B_1 contains a condition $\neg B_2$; or
- B_1 contains a condition $B_2 \Rightarrow B_3$, for some DRS B_3; or
- $B_1 \Rightarrow B_2$ is a DRS condition, in some DRS B_3; or
- B_1 contains a condition $B_2 \vee B_3$, for some DRS B_3; or
- $B_1 \vee B_2$ is a DRS condition, in some DRS B_3; or
- B_2 can access a DRS box B_3, and B_3 can access B_1.

For example, under those restrictions, the following DRS is not legal

(2.60)



because the condition *predicate* tries to access the discourse referent 'B' which however is not available under the accessibility restrictions. Intuitively, only upper-level boxes and boxes on the left are accessible.

2.5.3 Semantics as a mapping to first-order logic

We define the semantics of DRSs and DRS conditions by a mapping φ to first-order logic formulas as follows.

Discourse referents declared in the top-level DRS are mapped to existentially quantified variables and the list of conditions in the top-level DRS is mapped to a conjunction of their mappings.

$$\varphi\left(\begin{array}{|c|}\hline x_1 \ldots x_n \\ \hline C_1 \ldots C_m \\ \hline \end{array}\right) \equiv \exists x_1 \ldots \exists x_n [\varphi(C_1) \wedge \ldots \wedge \varphi(C_m)] \tag{2.61}$$

Every atomic condition maps to itself.

$$\varphi(R(x_1, \ldots, x_n)) \equiv R(x_1, \ldots, x_n) \tag{2.62}$$

Every negation-condition and disjunction-conditions is/are mapped by interpreting the negation and disjunction as the respective connectives in first-order logic.

$$\varphi(\neg B) \equiv \neg\varphi(B) \tag{2.63}$$

$$\varphi(B_1 \vee B_2) \equiv \varphi(B_1) \vee \varphi(B_2) \tag{2.64}$$

Finally, every implication-condition is mapped by interpreting the discourse referents of the *if*-box as universally quantified, and interpreting the implication as first-order logic implication.

$$\varphi(\boxed{\begin{array}{|c|}\hline x_1 \ldots x_n \\ \hline C_1 \ldots C_m \\ \hline\end{array}} \Rightarrow B) \equiv \forall x_1 \ldots \forall x_n [\varphi(C_1) \wedge \ldots \wedge \varphi(C_m) \Rightarrow \varphi(B)] \tag{2.65}$$

2.5.4 DRS as a meaning-representation of ACE sentences

The syntax of the DRS language used to represent ACE sentences [FKK07f] is somewhat restricted — the atomic conditions can be named only by a small set of names (*object*, *predicate*, *relation*, ...), and the discourse referents must be globally unique. In addition, there are several well-formedness constraints placed on the DRSs — every discourse referent that is declared must be used in the same DRS box, as an argument of at least one condition; every referent that is used in a condition must be declared (in an accessible DRS); predicate-conditions must always share discourse referents with object-conditions; certain conditions cannot share their first arguments, etc.

ACE texts are mapped into DRSs so that content words map to DRS atomic conditions, and function words to DRS connectives, arguments of conditions, or account for variable sharing. This mapping is largely based on a similar mapping in [BB99], although, in the case of ACE, the input language is more expressive and the resulting DRS is somewhat different. How this mapping is exactly performed, is beyond the scope of this thesis, here we only assume that the mapping reflects a possible (and common) English interpretation of the sentences.

The mapping of nouns, indefinite pronoun 'something', proper names, transitive verbs, copula verbs, and *of*-constructions is the following:

- countable common nouns map to object-conditions of the form *object(Ref, Lemma, countable, na, QType, QNum)*;
- the indefinite pronoun 'something' maps to an object-condition of the form *object(Ref, something, dom, na, na, na)*;
- proper names map to top-level object-conditions of the form *object(Ref, Lemma, named, na, eq, 1)*;
- transitive verbs map to predicate-conditions of the form *predicate(Ref, Lemma, SubjectRef, ObjectRef)*, containing information of the predicate-argument structure;

- copula forms map to object-conditions of the form *predicate(Ref, be, SubjectRef, ObjectRef)*;
- *of*-constructions map to relation-conditions of the form *relation(OwnedRef, of, OwnerRef)*;

where *Ref*, *SubjectRef*, *ObjectRef*, *OwnedRef*, *OwnerRef* are discourse referents, *Lemma* is the lemma form of the surface word form, *QNum* is a positive integer, and *QType* is one of *eq*, *geq*, *leq*, *greater*, *less*.

For example, the sentence

(2.66) Every friend of John owns at least 2 cars.

is represented in the Attempto DRS language as

(2.67)

A		
object(A, John, named, na, eq, 1)		
B	⇒	C D
object(B, friend, countable, na, eq, 1) relation(B, of, A)		object(C, car, countable, na, geq, 2) predicate(D, own, B, C)

where the object-condition is used for the common nouns 'friend' and 'cars' (the surface nouns are preserved in the lemmatized form); the object-condition is also used for the proper name 'John', in this case the object-condition is a top-level condition and has the 3rd argument *named*; the predicate-condition is used for the verb 'owns', due to the transitivity of which, the predicate-condition has arguments *B* and *C* that point to the object-conditions (which contain those discourse referents as first arguments); the relation-condition is used for the *of*-construction. The function word 'every' maps to an implication-condition but the word itself is not preserved in the DRS. The numerical quantifier 'at least 2' is stored as arguments *geq* and *2* in the object-condition.

In the Attempto DRS language, every atomic DRS condition also includes a sentence identifier, i.e. the conditions are wrapped into the binary term $-$ as $-(predicate(C,own,A,B),Id)$, where Id is the identifier of the sentence the condition originated from. As such, some information about the surface form of the text is preserved in the DRS and can be used e.g. in error messages by the parser. For simplicity, we omit the sentence identifiers in the DRS examples. The DRS representation of ACE texts and the ACE→DRS mapping is discussed in detail in [FKK07f].

The mapping of ACE texts to DRS which removes much of the arbitrary surface form of the texts, gives us a way to talk about the equivalence and difference of two ACE texts. We can say that two texts are equivalent (mean the same thing in ACE) if their DRSs are lexically identical. We can go even further and say that two texts are equivalent if the FOL representations of their DRSs are equivalent under the classical first-order logic semantics. Finally, we can define a number of background axioms (defined in FOL) that when added to the FOL representations make those representations logically equivalent. We now bring examples of those three levels of equivalence.

First level of equivalence (lexical-equivalence)

The mapping of ACE texts to DRSs is defined in such a way that many syntactically different texts get an identical DRS representation. For example

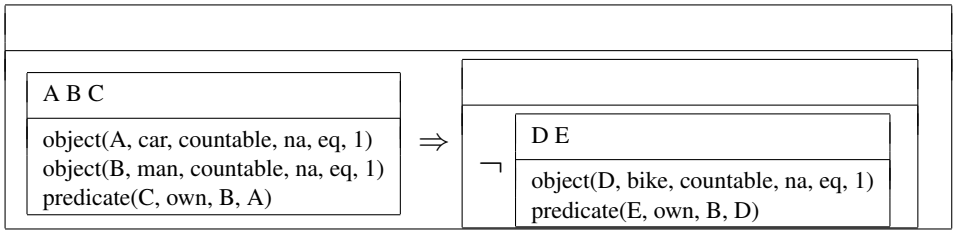
- both *if-then* sentences and *every*-sentences map to DRS implication-conditions;
- all forms of coordination (between sentences, relative clauses, and verb phrases) map to co-occurrence of conditions (in the case of 'and') or disjunction-conditions (in the case of 'or');
- all forms of negation map to the DRS negation-condition;
- Saxon genitives and explicit *of*-constructions both map to the relation-condition in the DRS;
- the difference between active and passive is lost on the DRS level.

For example, sentences 2.68–2.69 are equivalent as they have lexically identical DRSs (2.70).

(2.68) If a car is owned by a man then it is false that the man owns a bike.

(2.69) No man that owns a car owns a bike.

(2.70)



Second level of equivalence (FOL-equivalence)

There is also a second level of equivalence, i.e. the FOL level can declare two lexically different DRSs as logically equivalent using equivalence axioms, as for instance the elimination of double negation or De Morgan's rules. For example, sentences 2.71–2.72 are examples of equivalent sentences that do not have equivalent DRS representations. Still, standard first-order logic rewriting rules can transform their DRSs into a lexically identical form

(2.71) John does not like every dog.

(2.72) There is a dog. John does not like the dog.

as their meaning is given by either of the following equivalent FOL formulas. (Note that we use a shorter form of the formulas here, where the lemmas of nouns and verbs are used as predicate names and the proper names are represented as constants.)

$$\neg\forall X[dog(X) \Rightarrow like(John,X)] \tag{2.73}$$

$$\exists X[dog(X) \wedge \neg like(John,X)] \tag{2.74}$$

The following sentences are also equivalent

(2.75) John does not like a dog. (i.e. in standard English "John does not like any dog.")

(2.76) No dog is liked by John.

which can be shown by rewriting their corresponding FOL representations.

$$\neg\exists X[dog(X) \wedge like(John,X)] \equiv \tag{2.77}$$

$$\forall X[dog(X) \Rightarrow \neg like(John,X)] \tag{2.78}$$

For the same reason, the following texts are equivalent.

(2.79) Every first-guard is a guard that is not shielded by a guard.

(2.80) Every first-guard is a guard that no guard shields.

The following texts are equivalent.

(2.81) If somebody X writes something Y then X is an author and Y is a book.

(2.82) Everybody that writes something is an author. Everything that somebody writes is a book.

Again, this can be proved by examining their FOL representations.

$$\forall X,Y[write(X,Y) \Rightarrow author(X) \wedge book(Y)] \equiv \tag{2.83}$$

$$\forall X,Y[\neg write(X,Y) \vee author(X) \wedge book(Y)] \equiv \tag{2.84}$$

$$\forall X,Y[(\neg write(X,Y) \vee author(X)) \wedge (\neg write(X,Y) \vee book(Y))] \equiv \tag{2.85}$$

$$\forall X,Y[write(X,Y) \Rightarrow author(X)] \wedge \forall X,Y[write(X,Y) \Rightarrow book(Y)] \tag{2.86}$$

The following sentences have different DRS representations due to the fact that the DRS reflects to some extent the original sentence structure. Nevertheless, those sentences are equivalent in the first-order logic sense.

(2.87) It is false that there is a dog that is a cat.

(2.88) No dog is a cat.

(2.89) No cat is a dog.

Third level of equivalence (axiom-equivalence)

There is also a third level of equivalence — background axioms can be added to declare two DRSs equivalent even if they are not equivalent given the standard FOL semantics. For example, 'at least 2 cars' in the sentence

(2.90) Every man owns at least 2 cars.

is interpreted as

$$\forall A\ [object(A, man, countable, na, eq, 1) \Rightarrow \exists B, C\ [object(B, car, countable, na, geq, 2) \wedge predicate(C, own, A, B)]] \tag{2.91}$$

where the constants *geq* and *2* can be assigned meaning by the background axioms, so that 'at least 2' can be seen as equivalent to 'more than 1' in the reasoning.

Every ACE verb maps to a predicate-condition and every ACE noun to an object-condition. This is also the case for copula verb forms and indefinite pronouns 'something', 'everybody', etc. Their proper treatment during reasoning is assured by the following background axioms.

$$\forall X, Y[predicate(\_, be, X, Y) \Rightarrow X = Y] \tag{2.92}$$

$$\forall X[object(X, \_, \_, \_, \_, \_) \Rightarrow object(X, something, \_, \_, \_, \_)] \tag{2.93}$$

For example, let us look at the following semantically equivalent sentences

(2.94) If a man likes somebody that is a person then the person owns a car.

(2.95) If a man likes a person then the person owns a car.

where the second sentence lacks both the copula verb 'is' and the indefinite pronoun 'somebody', but should still be seen as equivalent to the first sentence.

Similarly, sentences 2.96–2.97 do not have lexically equivalent or FOL-equivalent DRSs but can still be considered logically equivalent.

(2.96) If a man owns a car X and owns a car Y then X is Y.

(2.97) Every man owns at most 1 car.

Such background axioms can be different for different applications of the DRS. In the following, we have chosen to interpret the word 'thing' as equivalent to 'something', and the words 'something' and 'somebody' as equivalent. Also, in our handling of the DRSs we treat plurals as distributive by default and no collective reading is supported (see [Sch04] for the interpretation where the plurals are collective by default). This means that we do not support sentences like:

(2.98) There are more than 3 cars.

(2.99) Mary's present is 3 flowers.

because the distributive reading can be applied only when the numerically quantified noun phrase is used as a subject or as an object of a transitive verb.

2.6 Syntactic sugar and paraphrasing

Syntactic sugar in ACE means that there are often different possibilities to reformulate an ACE text such as it is still interpreted in the same way. Syntactic sugar is offered on various language levels, starting with simple word-level where many function words have several equivalent spelling variants (e.g. 'every' = 'each', 'nobody' = 'no one'), and ending with the logical level where e.g. double negation is interpreted as the absence of negation. For example, anaphoric references to the same noun phrase can often be expressed by either a variable, pronoun, or a definite noun phrase; ACE coordination and negation can be applied on different phrasal levels, although semantically sentence coordination/negation would suffice; *if-then* sentences can be used instead of *every*-sentences; passive constructions can be avoided by using object relative clauses. Other than in many formal languages, in ACE, such syntactic sugar is not only tolerated but also carefully added to make the language easier to use — e.g. the user does not have to write ontological axioms in terms of *every*-sentences if he/she likes to think in terms of *if-then* sentences.

Another reason for offering various syntactic forms with the same meaning, is to allow for paraphrasing of ACE texts. Paraphrasing is essentially about reformulating the text so that the meaning is preserved, thus explaining the interpretation of the text better. For example,

(2.100) Every river flows-into a lake.

could be paraphrased as

(2.101) If there is a river then there is a lake such that the river flows-into the lake.

to make it absolutely clear that the existence of a river and a lake is not guaranteed by this constraint. Such paraphrasing would not be possible in case the language offered no syntactic sugar. The ACE tools include two ways to paraphrase ACE texts — either into the Core ACE subset [FKS05] that uses only sentence coordination and negation, or the NP ACE subset [FKK06a] that tries to use noun phrases with compact relative clauses. Both paraphrasers are actually implemented as DRS verbalizers.

2.7 Other versions of controlled English

In this section, we discuss other existing controlled natural languages (which all happen to be fragments of English). We only focus on more recent languages and languages

currently under active development. Also, we mainly discuss the features and design decisions which make ACE unique as compared to other controlled natural languages.

Expressive and recently developed versions of controlled English include PENG [Sch05b, Sch05a, ST04, ST06], Common Logic Controlled English [Sow04], Boeing's Computer Processable Language [CHJ+05], and E2V [PH03]. For an exhaustive list of controlled natural languages see [Poo06] which lists 32 languages altogether.

2.7.1 Processable English (PENG)

Rolf Schwitter's PENG[3] branched out from the research done on ACE. Therefore the designs of the two languages are quite similar. In recent years, more features have been added to ACE, making it both syntactically and semantically more expressive than PENG. Such features are e.g. support for plural noun phrases with numerical quantifiers, complex noun phrases as anaphors and antecedents, numbers and strings in noun phrase position, modality ('can' and 'must'), *that*-subordination. In addition to first-order logic reasoning which both PENG and ACE offer, the ACE tools also include a parser with large lexicon and unknown word guessing support, and a paraphraser that reformulates the input sentence in a syntactically different way.

Research on PENG, on the other hand, stresses the need for syntax-aware editing tools for controlled natural languages, and has focused on the development of ECOLE [SLH03], a look-ahead text editor that guides the user in constructing only syntactically acceptable PENG sentences. Recently, PENG has been used as a starting point of developing a new OWL-compatible controlled natural language Sydney OWL Syntax (discussed in section 4.3.1).

2.7.2 Common Logic Controlled English (CLCE)

Another ACE-like controlled English is John Sowa's CLCE [Sow04, Sow07], which has been designed as a "human interface" to the ISO standard Common Logic[4]. However, there is only a partial specification available, and a parser for CLCE has not been published.

The main difference between CLCE and ACE is that CLCE supports four kinds of common nouns: categorial nouns (nouns which specify a category, e.g. 'dog'), relational nouns (nouns which specify a dyadic relation to some entity, e.g. 'child'), functional nouns (nouns which specify a functional dyadic relation to some entity, e.g. 'mother'), and variadic relational or functional nouns (nouns which specify a relation to more than one entities, e.g. 'set'). All these nouns can be used as categorial, the relational reading is trigged syntactically by using the preposition *of*. Words and their lexical classes have to be declared within the CLCE text. For example, the variadic functional noun 'set' can be declared as

(2.102) Declare set as functional noun (x1 set of x2 ...).

[3] http://www.ics.mq.edu.au/~rolfs/peng/

[4] http://cl.tamu.edu/

and then used as

(2.103) John owns the set of (a cat, a dog, and an elephant).

In ACE, there is just one common noun category but as ACE supports *of*-constructions, sentences which are syntactically CLCE can be written in ACE. For example, in ACE, one can write

(2.104) The mother of Sue is Mary.

and recover from the presence of the preposition *of* that the noun 'mother' is relational. However, the functionality of this relation is left underspecified in the ACE text (and its DRS representation).

2.7.3 Computer Processable Language (CPL)

CPL [CHJ+05] is a controlled English developed at Boeing, and used experimentally for various purposes, e.g. to annotate video clips to support semantic search, and to rephrase texts on chemistry. CPL is closer to everyday English than ACE, PENG, or CLCE are, in the sense that it uses fewer strict rules and its interpreter is expected to "smartly" resolve various ambiguities. For example, while in ACE all prepositional phrases (with the exception of *of*-phrases) attach to verbs, in CPL they can either attach to nouns or verbs, and the interpreter is expected to make a resolution that is acceptable in most cases for the author of the text. The interpreter also handles nominalizations (nominalized verbs are mapped to their verbal counterparts) and guesses the word senses of nouns and verbs with the help of the WordNet[5] lexicon. Errors are handled by sophisticated error resolution, i.e. in addition to grammar rules that detect and handle legal CPL sentences, rules have also been written to detect most common errors. The result of parsing is a logical formula in a frame-based knowledge representation language Knowledge Machine [CP04] on which a reasoner can be applied.

2.7.4 E2V and other fragments of English

E2V [PH03] is a fragment of English that corresponds to the decidable two-variable fragment of first-order logic ($\mathcal{L}^2$). E2V includes determiners ('every', 'no', 'a'), common nouns, transitive verbs, relative clauses, reflexives (e.g. 'himself'), and pronouns (e.g. 'him'). An example of an E2V sentence is

(2.105) Every artist who employs a carpenter despises every beekeeper who admires him.

Syntactically, E2V is a subset of ACE. However, its treatment of pronominal references is different — pronouns always refer to the closest noun in the syntax tree, and not to the closest noun in the surface order of words. This makes a difference in case the

[5] `http://wordnet.princeton.edu`

preceding noun phrase contains a relative clause. This different treatment seems to be mainly motivated by better reasoning properties over the target logical form — with an ACE-like treatment, a mapping to $\mathcal{L}^2$ would not be possible.

[PH04, PHT06, Thi06] discuss more fragments of English which extend E2V. In general, E2V, as well as its extensions have been developed to study the computational properties of certain linguistic structures (e.g. relative clauses, ditransitive verbs, anaphoric references). The intention of the authors has not been to develop a real-world knowledge representation language by adding features which would increase the usability of the language. Instead, only language features that introduce interesting computational problems have been added. For example, disjunction, adverbs, intransitive verbs, passive constructions, and *of*-constructions could been seen as features that would add usability but that would preserve the expressivity of the target logical language. While E2V has nice computational properties, some of its extensions that use e.g. a general (co-indexing-based) reference mechanism (instead of pronominal references) become computationally undecidable.

Chapter 3

Web Ontology Language OWL

3.1 Introduction

In this chapter, we introduce the Web Ontology Language (OWL). OWL (i.e. its three species OWL Lite, OWL DL, and OWL Full) has been a W3C standard since 2004. At the moment, an update to this standard is being finalized. This update is preliminarily called OWL 1.1. In this chapter, we discuss only OWL 1.1, and often briefly say OWL. As OWL 1.1 is only a minor update, and furthermore backwards compatible, a lot of statements made in this chapter apply also to the current OWL. Furthermore, some documents of the current OWL specification have not received an update yet (e.g. the "Requirements" document [Hef04]). As they are still relevant for our discussion we will refer to them.

The OWL 1.1 specification[1] is (currently) divided into the following documents:

- OWL 1.1 Web Ontology Language Overview
- OWL 1.1 Web Ontology Language Structural Specification and Functional-Style Syntax
- OWL 1.1 Web Ontology Language Mapping to RDF Graphs
- OWL 1.1 Web Ontology Language Model-Theoretic Semantic
- OWL 1.1 Web Ontology Language Tractable Fragments
- OWL 1.1 Web Ontology Language XML Syntax

Three of those specifications (Structural Specification and Functional-Style Syntax, Mapping to RDF Graphs, XML Syntax) are concerned with various syntaxes for OWL. All those syntaxes are bidirectionally mappable to each other, and the need for RDF and XML based syntaxes is mainly motivated by the current structure of the Semantic Web stack of languages. In the following, we only deal with the Functional-Style syntax. The

[1] `http://webont.org/owl/1.1/`

"Tractable Fragments" document [Gra07] discusses various subsets (i.e. semantically less expressive fragments) of OWL 1.1 for which the reasoning tasks are known to have better computational complexity. In the following, we are not going to discuss those fragments. The "Model-Theoretic Semantics" document gives semantics to OWL 1.1 constructs, using the Functional-Style Syntax.

In addition to the aspects of OWL that are being standardized, there are several other important and interesting aspects that, in the following, concern us. How is OWL being used? With which editing tools? With which alternative syntaxes? What kind of problems do users encounter when working with OWL?

For all of those questions, we are interested in how (controlled) natural language based solutions can help to solve the existing problems or enhance the existing solutions to those problems.

3.2 Syntax and semantics of OWL

OWL 1.1 is based on the description logic $\mathcal{SROIQ}$ [HKS06] which is extended with datatypes and data properties, and a form of meta-modeling called punning (see [GHP+06]). Here we provide the syntax and semantics for $\mathcal{SROIQ}$, and discuss data properties later as part of the extensions in section 6.3. Punning means that names can be used for several purposes, e.g. *person* can be used as the name of a class and the name of an individual at the same time. From a semantic point of view, these names can be seen be different names (i.e. *person-the-class* and *person-the-individual*).

3.2.1 Syntax

Every $\mathcal{SROIQ}$ property is either a named property (denoted R) or the inverse of a named property (denoted R^-).

Every $\mathcal{SROIQ}$ class is a named class, is of the form $\top$, or is a complex class description. Every complex class description is of one of the following forms: $\neg C$, $C \sqcap D$, $C \sqcup D$, $\forall RC$, $\exists RC$, $\exists R\,\text{SELF}$, $\leq nRC$, $\geq nRC$, $\{a_1 \ldots a_m\}$, where C and D are classes, $a_1 \ldots a_m$ are individuals, R is a property, n is a non-negative integer, and m is a positive integer. Classes that involve properties are called property restrictions. Those restrictions that contain $\leq n$ or $\geq n$ are called cardinality restrictions.

Every $\mathcal{SROIQ}$ general class inclusion (GCI) axiom (*SubClassOf*-axiom) has the form $C \sqsubseteq D$, where C and D are classes. The form $C \doteq D$ is used as a short-hand for the combination of $C \sqsubseteq D$ and $D \sqsubseteq C$.

Every $\mathcal{SROIQ}$ property axiom has the form $R_1 \circ \ldots \circ R_n \sqsubseteq S$ or has the form $Dis(R,S)$, where R_1, ..., R_n, S, R are properties. Expressions of the form $R_1 \circ \ldots \circ R_n$ are called property chains.

Every $\mathcal{SROIQ}$ individual assertion axiom (or fact) is of one of the following forms: $a : C$, $(a,b) : R$, $(a,b) : \neg R$, $a = b$, $a \neq b$, where a and b are individuals, C is a class, and R a property.

Every $\mathcal{SROIQ}$ ontology is a set of $\mathcal{SROIQ}$ axioms.

3.2.2 Semantics

The following mapping table gives the semantics of $\mathcal{SROIQ}$ in first-order logic. We map each class C to a FOL formula $\varphi(C,x)$ with one free variable x, and each property R to a binary predicate $\varphi(R,x,y)$ with two free variables x and y, in the following way

$$\varphi(\top,x) \equiv \top \tag{3.1}$$

$$\varphi(A,x) \equiv A(x) \tag{3.2}$$

$$\varphi(\neg C,x) \equiv \neg\varphi(C,x) \tag{3.3}$$

$$\varphi(C \sqcap D,x) \equiv \varphi(C,x) \wedge \varphi(D,x) \tag{3.4}$$

$$\varphi(C \sqcup D,x) \equiv \varphi(C,x) \vee \varphi(D,x) \tag{3.5}$$

$$\varphi(\forall RC,x) \equiv \forall y[\varphi(R,x,y) \Rightarrow \varphi(C,y)] \tag{3.6}$$

$$\varphi(\exists RC,x) \equiv \exists y[\varphi(R,x,y) \wedge \varphi(C,y)] \tag{3.7}$$

$$\varphi(\exists R\,\mathrm{SELF},x) \equiv \varphi(R,x,x) \tag{3.8}$$

$$\varphi(\leq nRC,x) \equiv \forall y_1 \ldots y_{n+1}[\bigwedge_{i=1}^{n+1}(\varphi(R,x,y_n) \wedge \varphi(C,y_n)) \Rightarrow \bigvee_{i=1,j=i+1}^{n+1}(y_i = y_j)] \tag{3.9}$$

$$\varphi(\geq nRC,x) \equiv \forall y_1 \ldots y_n[\bigwedge_{i=1}^{n}(\varphi(R,x,y_n) \wedge \varphi(C,y_n)) \wedge \bigwedge_{i=1,j=i+1}^{n}\neg(y_i = y_j)] \tag{3.10}$$

$$\varphi(\{a_1 \ldots a_n\},x) \equiv \bigvee_{i=1}^{n}(x = a_i) \tag{3.11}$$

$$\varphi(R,x,y) \equiv R(x,y) \tag{3.12}$$

$$\varphi(R^-,x,y) \equiv R(y,x) \tag{3.13}$$

where C and D are class descriptions and A is a named class; R is a property description (either in the form R or R^-); and $a_1, \ldots, a_n$ are named individuals.

SubClassOf-axioms map to universally quantified implications where the *if*-part and *then*-part share a variable.

$$C \sqsubseteq D \equiv \forall x[\varphi(C,x) \Rightarrow \varphi(D,x)] \tag{3.14}$$

$$C \doteq D \equiv \forall x[\varphi(C,x) \Leftrightarrow \varphi(D,x)] \tag{3.15}$$

Note that the classes C and D in $C \doteq D$ are often called defined or complete, while C in $C \sqsubseteq D$ is called primitive or partial.

Property axioms are mapped in the following way

$$R_1 \circ \ldots \circ R_n \sqsubseteq S \equiv \forall x_1 \ldots x_n [\varphi(R_1, x_1, x_2) \wedge \ldots \wedge \varphi(R_n, x_n, x_{n+1}) \Rightarrow \varphi(S, x_1, x_{n+1})] \tag{3.16}$$

$$Dis(R,S) \equiv \forall x, y [\varphi(R, x, y) \Rightarrow \neg\varphi(S, x, y)] \tag{3.17}$$

where S, R, R_1, ..., R_n are property descriptions. Finally, facts are mapped in the following way

$$a : C \equiv \varphi(C, a) \tag{3.18}$$

$$(a,b) : R \equiv \varphi(R, a, b) \tag{3.19}$$

$$(a,b) : \neg R \equiv \neg\varphi(R, a, b) \tag{3.20}$$

$$a = b \equiv a = b \tag{3.21}$$

$$a \neq b \equiv \neg(a = b) \tag{3.22}$$

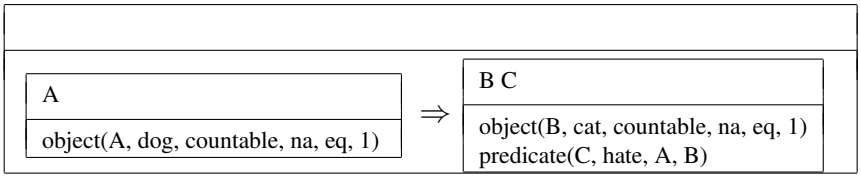
where C is a class description; R is a property description; and a, b are named individuals. Note that $\mathcal{SROIQ}$ does not make the so-called unique name assumption — differently named individuals can be asserted to be equivalent or different, but in the absence of such assertions, the relationship between two individuals is unknown (or possibly implicit). Thanks to the powerful class description constructor $\{a_1 \ldots a_n\}$, facts 3.18–3.22 can be written as semantically equivalent *SubClassOf*-axioms: $\{a\} \sqsubseteq C$, $\{a\} \sqsubseteq \exists R\, \{b\}$, $\{a\} \sqsubseteq \neg(\exists R\, \{b\})$, $\{a\} \sqsubseteq \{b\}$, $\{a\} \sqsubseteq \neg\{b\}$.

The mapping to first-order logic makes it easier to compare OWL axioms to ACE sentences. For example, the axiom $dog \sqsubseteq \exists\, hate\, cat$ maps to the FOL-formula

$$\forall x\, [dog(x) \Rightarrow \exists y (cat(y) \wedge hate(x,y))] \tag{3.23}$$

Similarly, the ACE sentence "Every dog hates a cat." which has the DRS-representation

(3.24)



is mapped by the standard DRS→FOL transformation into something very similar.

$$\forall A\, [object(A, dog, countable, na, eq, 1) \Rightarrow \exists B, C\, (object(B, cat, countable, na, eq, 1) \wedge predicate(C, hate, A, B))] \tag{3.25}$$

3.2.3 Non-structural restrictions on axioms

$\mathcal{SROIQ}$, as well as OWL, includes a set of non-structural restrictions on axioms [MPSH07]. Those restrictions are essentially rules that remove some (otherwise) syntactically legal constructions from the language. For example, properties that have been defined via property chains cannot be used in cardinality restrictions. Those rules achieve that certain reasoning tasks on OWL ontologies remain decidable.

3.2.4 Meta-statements

OWL also supports meta-statements so that users can e.g. label and annotate axioms and entities, express version information, import other ontologies, and deprecate some statements, all in the same ontology language. Such constructs are needed in real-world ontology engineering, as they simplify e.g. sharing the ontologies and displaying them in ontology editors.

3.3 Various alternative OWL syntaxes

Several alternative syntaxes have been developed for OWL in order to account for different usages and requirements. RDF syntaxes make OWL compatible with the Semantic Web stack, XML syntaxes enable OWL to be processed by XML tools (e.g. those that implement XPath [CD99] or XSLT [Cla99]). Compact syntaxes are needed to present snippets of OWL ontologies in scientific papers or email discussions, some other syntaxes fit better with certain programming languages (e.g. Prolog, Lisp, Javascript), etc. In this section, we look at the main alternative syntaxes that an OWL user can choose from.

3.3.1 OWL 1.1 Functional-Style Syntax

OWL 1.1 Functional-Style Syntax [MPSH07] can be seen as a direct mapping of the $\mathcal{SROIQ}$ syntax to a functional notation, with an addition of some short-hand constructs. We present it here as later in the thesis we are going to use it in our Prolog coding of the various translations of OWL.

In the functional syntax, inverse properties R^- are written as *InverseObjectProperty*(R), where R is a named property. The notation for class descriptions is presented in table 3.1 and the notation for axioms in table 3.2.

To gain a shorter notation and to be compatible with existing examples, in the following, we sometimes write *domain* to mean *ObjectPropertyDomain*, *range* to mean *ObjectPropertyRange*, *functional* to mean *ObjectFunctionalProperty*, *sameAs* to mean *SameIndividual*, and *differentFrom* to mean *DifferentIndividuals*.

OWL class in functional notation	OWL class in DL-notation
owl:Thing	$\top$
owl:Nothing	$\neg\top$
ObjectComplementOf(C)	$\neg C$
ObjectIntersectionOf($C_1 \ldots C_n$)	$C_1 \sqcap \ldots \sqcap C_n$
ObjectUnionOf($C_1 \ldots C_n$)	$C_1 \sqcup \ldots \sqcup C_n$
ObjectOneOf($a_1 \ldots a_n$)	$\{a_1 \ldots a_n\}$
ObjectSomeValuesFrom($R\ C$)	$\exists\, R\ C$
ObjectAllValuesFrom($R\ C$)	$\forall\, R\ C$
ObjectHasValue($R\ a$)	$\exists\, R\ \{a\}$
ObjectExistsSelf(R)	$\exists\, R$ SELF
ObjectMinCardinality($n\ R\ C$)	$\geq\ n\ R\ C$
ObjectMaxCardinality($n\ R\ C$)	$\leq\ n\ R\ C$
ObjectExactCardinality($n\ R\ C$)	$(\leq\ n\ R\ C) \sqcap (\geq\ n\ R\ C)$
ObjectMinCardinality($n\ R$)	$\geq\ n\ R\ \top$
ObjectMaxCardinality($n\ R$)	$\leq\ n\ R\ \top$
ObjectExactCardinality($n\ R$)	$(\leq\ n\ R\ \top) \sqcap (\geq\ n\ R\ \top)$

Table 3.1: Relating OWL 1.1 classes in Functional-Style Syntax with DL-notation. C, C_1, $\ldots$, C_n are class descriptions; R is a property description; and a, a_1, $\ldots$, a_n are named individuals.

3.3.2 RDF-based syntaxes

RDF/XML-based syntax[2] is the normative syntax for the current version of OWL. Most OWL tools support this syntax, and as it encodes RDF, also general RDF tools can process it to some extent even if they do not fully support (the semantics of) OWL.

OWL can also be expressed in any RDF serialization syntax: Notation 3[3], Turtle[4], etc. While some of these syntaxes are more readable than RDF/XML, all of them share the problem that they have no special knowledge of OWL's constructs, i.e. they see all statements as RDF triples. Thus, OWL ontologies expressed in RDF come out too verbose and unnecessarily hard to read. The following example of Notation 3 expresses the OWL axiom $man \sqsubseteq \exists\, own\ car$.

```
_:bnode0 rdf:type owl:Restriction ;
    owl:onProperty <http://example.org/ontology#own> ;
    owl:someValuesFrom <http://example.org/ontology#car> .
<http://example.org/ontology#man> rdfs:subClassOf _:bnode0 .
```

Because in RDF, everything is either a node or a property between two nodes, an anonymous node (so-called blank node) :_bnode0 must be used to link the class name

[2] http://www.w3.org/TR/owl-ref/

[3] http://www.w3.org/DesignIssues/Notation3

[4] http://www.dajobe.org/2004/01/turtle/

OWL axiom in functional notation	OWL axiom in DL-notation
SubClassOf(C D)	$C \sqsubseteq D$
EquivalentClasses($C_1 \ldots C_n$)	$C_i \doteq C_j$, for each $i, j \in \{1,\ldots,n\}, i \neq j$
DisjointClasses($C_1 \ldots C_n$)	$C_i \sqsubseteq \neg C_j$, for each $i, j \in \{1,\ldots,n\}, i \neq j$
DisjointUnion(D $C_1 \ldots C_n$)	DisjointClasses($C_1 \ldots C_n$) and EquivalentClasses(D ObjectUnionOf($C_1 \ldots C_n$))
SubObjectPropertyOf(R S)	$R \sqsubseteq S$
SubObjectPropertyOf(SubObjectPropertyChain($R_1 \ldots R_n$) S)	$R_1 \circ \ldots \circ R_n \sqsubseteq S$
EquivalentObjectProperties($R_1 \ldots R_n$)	$R_i \sqsubseteq R_j$ and $R_j \sqsubseteq R_i$, for each $i, j \in \{1,\ldots,n\}, i \neq j$
DisjointObjectProperties($R_1 \ldots R_n$)	$Dis(R_i, R_j)$, for each $i, j \in \{1,\ldots,n\}, i \neq j$
ObjectPropertyDomain(R C)	$\exists R \top \sqsubseteq C$
ObjectPropertyRange(R C)	$\exists R^- \top \sqsubseteq C$
InverseObjectProperties(R S)	$R \sqsubseteq S^-$ and $S^- \sqsubseteq R$
FunctionalObjectProperty(R)	$\top \sqsubseteq \leq 1 R \top$
InverseFunctionalObjectProperty(R)	$\top \sqsubseteq \leq 1 R^- \top$
ReflexiveObjectProperty(R)	$\top \sqsubseteq \exists\, R$ SELF
IrreflexiveObjectProperty(R)	$\top \sqsubseteq \neg(\exists\, R$ SELF$)$
SymmetricObjectProperty(R)	$R \sqsubseteq R^-$
AsymmetricObjectProperty(R)	$Dis(R, R^-)$
TransitiveObjectProperty(R)	$R \circ R \sqsubseteq R$
ClassAssertion(a C)	$a : C$
ObjectPropertyAssertion(R a b)	$(a,b) : R$
NegativeObjectPropertyAssertion(R a b)	$(a,b) : \neg R$
SameIndividual($a_1 \ldots a_n$)	$a_i = a_j$, for each $i, j \in \{1,\ldots,n\}, i \neq j$
DifferentIndividuals($a_1 \ldots a_n$)	$a_i \neq a_j$, for each $i, j \in \{1,\ldots,n\}, i \neq j$

Table 3.2: Relating axioms in OWL 1.1 Functional-Style Syntax with DL-notation. C, D, $C_1, \ldots, C_n$ are class descriptions; S, R, $R_1, \ldots, R_n$ are property descriptions; and a, b, $a_1, \ldots, a_n$ are named individuals.

man to the complex (and anonymous) class description $\exists$ *own car*.

3.3.3 OWL 1.1 XML-based syntax

The OWL 1.1 XML-based (and at the same time non-RDF) syntax specified in [GMPS07] will become the normative syntax for OWL 1.1. The XML-based syntax achieves compatibility with XML tools and processing technologies such as XSLT which are not entirely compatible with RDF (even if it is expressed in XML) due to its commitment to the triple-based data model. As the XML-syntax is defined by a direct mapping from the Functional-Style syntax, it is equally simple (or hard) to read to the Functional-Style syntax. The following example expresses again $man \sqsubseteq \exists\, own\, car$, now in the XML-syntax.

```
<SubClassOf>
    <OWLClass URI="http://example.org/ontology#man"/>
    <ObjectSomeValuesFrom>
        <ObjectProperty URI="http://example.org/ontology#own"/>
        <OWLClass URI="http://example.org/ontology#car"/>
    </ObjectSomeValuesFrom>
</SubClassOf>
```

3.3.4 Description Logics' syntax

As OWL is directly related to description logics (DL), it is very common to use a DL-notation in papers about OWL. Also, most OWL editors support DL-notation as means for the user to enter complex class descriptions. We used this notation when giving the syntax and semantics for $\mathcal{SROIQ}$, and will use this notation throughout this thesis as it is more concise.

3.3.5 Manchester OWL Syntax

Manchester OWL Syntax[5] [HDG+06] can been seen as an improvement of the DL-notation, especially because this syntax is influenced by the complaints of OWL users who had difficulties reading and writing in the DL-notation.

Manchester Syntax is primarily meant for entering (complex) class descriptions in OWL editors. Manchester Syntax replaces logic symbols ($\exists$, $\forall$, $\sqcap$, $\sqcup$, $\neg$, $\in$, $\geq$, $\leq$, $=$) by English keywords ('some', 'only', 'and'/'that', 'or', 'not', 'has', 'min', 'max', 'exactly') and uses infix notation for property restrictions (e.g. 'own some car' instead of $\exists\, own\, car$) to make class descriptions more like natural language noun phrases. E.g. the class description

(3.26) Person **and** hasChild **some** (Person **and** (hasChild **only** Man) **and** (hasChild **some** Person))

[5] http://www.co-ode.org/resources/reference/manchester_syntax/

describes the class of people who have at least one child that has some children that are only men (i.e. grandparents that only have grandsons). Note that brackets should be used to disambiguate the meaning of the expression.

Although, the Manchester Syntax is primarily meant for class descriptions, it can be used for general axioms as well. In this case, standard OWL keywords (e.g. *SubClassOf*, *DisjointClasses*) are used. An example of a *SubClassOf*-axiom is

(3.27) Person **and** hasChild **some** Person **subClassOf** ((hasChild **only** Man) **and** (hasChild **some** Person))

Note that in ontology editors that use Manchester Syntax, the class descriptions can be made easily more readable by keyword highlighting and syntax-aware indentation.

While an improvement over DL-notation, the lack of determiners and specifically the heavy use of parentheses render Manchester Syntax unnatural in comparison to normal English.

3.3.6 Structured Ontology Format

[She07] defines the Structured Ontology Format (SOF), a simple textual format to express OWL 1.1 ontologies. The format makes it easier to manipulate ontologies with text tools such as *grep* and import into the native data structures of programming languages such as Javascript. Class descriptions are written in Extended Manchester OWL Syntax (EMOS). The following figure shows an example of a SOF-formatted ontology.

```
namespaces:
    "" : http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#
    food : http://www.w3.org/TR/2003/PR-owl-guide-20031209/food#

classes:
    Wine:
        subsumed by:
            - food:PotableLiquid # note use of namespace
            - hasMaker exactly 1 # class descriptions use EMOS
            - locatedIn some Region
    TableWine:
         equivalent to:
            - Wine that hasSugar value Dry

properties:
    hasMaker:
         inverses: [ producesWine ] # bracketed syntax for sequences
         functional: # some keys don't need values
    locatedIn:
         range: [ Region ]
         transitive:

individuals:
    StonleighSauvignonBlanc:
         member of:
```

```
    - Wine
related:
    hasSugar: [ Dry ] # sequences can be bracketed
    hasMaker:
        - Stonleigh # or use a line-oriented style.
```

3.4 OWL editors

In order to work with OWL ontologies, users are often encouraged to use front-end tools. Such tools — TopBraid Composer[6], Protégé[7], SWOOP[8], OntoStudio[9] — are not simple text editors that only understand the OWL syntaxes, rather, they offer an elaborate graphical front-end with forms, trees, wizards, etc. and try to hide the OWL syntax. For complex class descriptions, however, they revert to using the syntax of description logics or (more recently) Manchester OWL Syntax, and thus fail to hide the complexities of OWL in general. (See figure 3.1.) They also restrict the user in various ways, for example the names of classes/properties/individuals have to be declared before they can be used, and entering *SubClassOf*-axioms with a complex left-side is made hard or is even impossible in most tools.

[DMB+06] compares TopBraid Composer and Protégé (version 3.x) and finds several problems that both novices and experts encountered: it was hard to get an overview of the usage of classes/properties, ontology visualization was not helpful, DL-notation which was prominently featured in both editors was confusing, etc. The authors' general conclusion was that current tools have acceptable quality when given to experts in logic but have a steep learning curve with newcomers to ontology building. Therefore, alternative methods for working with ontologies are clearly needed.

One of the benefits of using an OWL editor, is the seamless access to OWL reasoning tools such as Pellet[10], FaCT++[11], RacerPro[12], KAON2[13]. The main function of these reasoners is to make implicit *SubClassOf*-axioms explicit, and also to explain the reasons for those derivations, or in case the ontology is inconsistent, provide (precise) reasons for the inconsistency.

3.5 Natural language can eliminate problems that one encounters when using OWL

[RDH+04] provides a detailed list of problems and pitfalls that users encounter when working with OWL, and expresses the need for a "pedantic but explicit" paraphrase

[6] http://www.topbraidcomposer.com
[7] http://protege.stanford.edu
[8] http://www.mindswap.org/2004/SWOOP/
[9] http://www.ontoprise.de
[10] http://pellet.owldl.com/
[11] http://owl.man.ac.uk/factplusplus/
[12] http://www.racer-systems.com/
[13] http://kaon2.semanticweb.org/

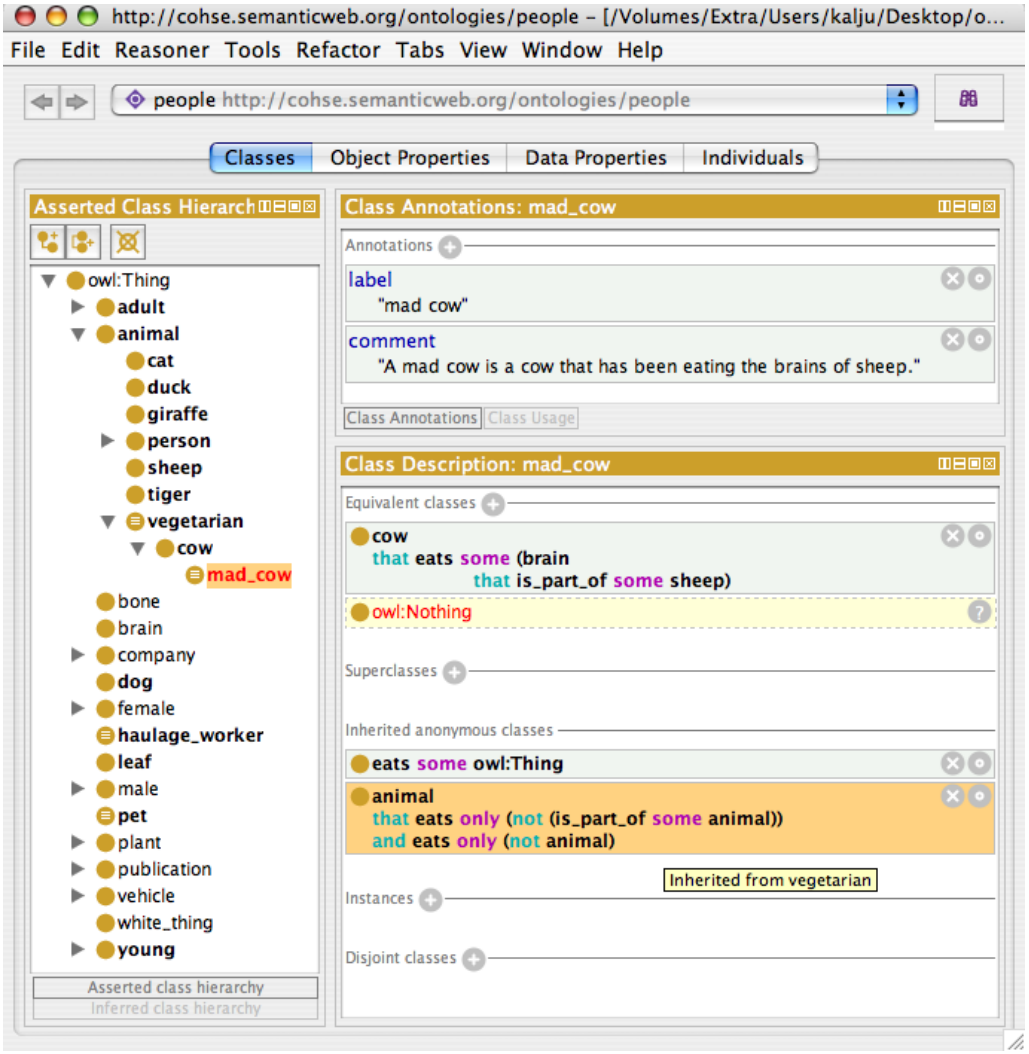


Figure 3.1: Screenshot of the "Classes tab" in Protégé 4. The tab is divided into three windows — the left window displays the *SubClassOf*-hierarchy of named classes, the upper right window shows human-readable annotations of the class *mad_cow*, and the lower right window shows complex class descriptions in Manchester OWL Syntax. An integrated OWL reasoner has detected that the class *mad_cow* is equivalent to *owl:Nothing*, i.e. that it is unsatisfiable.

language for OWL expressions. [KPSG06] brings more concrete examples of similar pitfalls.

The most common problems listed in [RDH+04] are:

1. failure to make all information explicit — assuming that information implicit in names is "represented" and available to the classifier;
2. mistaken use of universal (i.e. *ObjectAllValuesFrom*) rather than existential restrictions (i.e. *ObjectSomeValuesFrom*) as the default;
3. open world reasoning (the authors call it "the biggest single hurdle");
4. the effect of range and domain constraints as axioms (the authors call it "the largest single source of errors after the open world reasoning");
5. trivial satisfiability of universal restrictions — that "only" (i.e. *ObjectAllValuesFrom*) does not imply "some" (i.e. *ObjectSomeValuesFrom*), i.e. $\forall\, R\, C \not\sqsubseteq \exists\, R\, C$;
6. the difference between defined and primitive classes and the mechanics of converting one to the other;
7. difference between the linguistic and logical usage of "and" and "or";
8. easy-to-confuse representations of "some not", i.e. $\exists R(\neg C)$ and "not some", i.e. $\neg(\exists RC)$;
9. expecting classes to be disjoint by default (this is essentially a subproblem of problem 1); and
10. the difficulty of understanding subclass axioms used for implication.

While some of those problems, e.g. "open world reasoning" and "disjointness by default", are likely to be shared by all possible syntaxes for OWL, several of the problems, e.g. confusion about domains and ranges, and *ObjectAllValuesFrom* vs *ObjectSomeValuesFrom* are largely specific to the DL-notation and similar OWL syntaxes that the authors had experience with. For example, an easy solution for the confusion about *ObjectAllValuesFrom* would be to remove this construct from the language. Note that this would not reduce the semantic expressivity of the language, as $\forall RC$ can be expressed by $\neg\exists R(\neg C)$.

The concrete pitfalls discussed in [KPSG06] are:

- $A \doteq C$ was used, but $A \sqsubseteq C$ was meant;
- $A \sqsubseteq C; A \sqsubseteq D$ was used, but $A \sqsubseteq C \sqcup D$ was meant;
- $domain(P,A); range(P,B)$ was used, but $A \sqsubseteq \forall\, P\, B$ was meant;
- $domain(P,A); domain(P,B)$ was used, but $domain(P, A \sqcup B)$ was meant.

Note that all those pitfalls are easily avoided if one realizes the natural language representation of the formulas involved, namely (note that we use English words like 'human' and 'river' in order to build more meaningful sentences):

(3.28) Every man is a human and every human is a man. $\neq$ Every man is a human.

(3.29) Every human is a man and every human is a woman. $\neq$ Every human is a man or is a woman.

(3.30) Everything that flows-into something is a river. Everything that something flows-into is a lake. $\neq$ Everything that a river flows-into is a lake.

(3.31) Everything that flows-into something is a river. Everything that flows-into something is a stream. $\neq$ Everything that flows-into something is a river or is a stream.

Another mistake that users often make is that they think that *SubClassOf* can be used to encode the "part-whole" relationship [WCH87]. E.g. the users interpret $town \sqsubseteq country$ as meaning that every town is part of a country. In natural language the distinction is usually clear, e.g. the axioms

(3.32) $$town \sqsubseteq country$$

(3.33) $$town \sqsubseteq \exists\, is\_part\_of\ country$$

are expressed differently by either using a copula verb verb 'be', or a normal transitive verb (e.g. 'contain') or a dedicated construction like 'is a part of'.

(3.34) Every town is a country.

(3.35) Every town is contained by a country. (Every town is a part of a country.)

where the statement 3.34 is clearly false.

[HG07] discusses a common mistake related to the use of inverses and reciprocal statements. While for individual assertion axioms it always holds that one can reverse the sides of the individuals if one inverts the property, e.g.

(3.36) $$\{John\} \sqsubseteq \exists\, love\ \{Mary\} \quad \equiv \quad \{Mary\} \sqsubseteq \exists\, love^{-}\ \{John\}$$

it is not the case for class descriptions in general

(3.37) $$car \sqsubseteq \exists\, contain\ engine \quad \not\equiv \quad engine \sqsubseteq \exists\, contain^{-}\ car$$

Again, rephrasing those axioms in natural language can explain better their differences.

(3.38) John loves Mary. = Mary is loved by John.

(3.39) Every car contains a engine. $\neq$ Every engine is contained by a car.

It thus seems that many problems that the OWL users face when trying to author ontologies or read ontologies built by others are rooted in the OWL syntax. Only very simple ontologies can be understood and edited in visual terms, as trees or graphs of *SubClassOf*-relations. For more complex ontologies, a formal-looking syntax is currently exposed to the user.

It also seems that many problems can be avoided by adopting a syntax which is closer to the way how people think, i.e. a natural language based syntax. In the next chapter on related work (chapter 4), we look at existing efforts in trying to design such a syntax and, after that, we describe our own approach that is based on Attempto Controlled English.

Chapter 4

Related work

4.1 Introduction

In this chapter, we discuss the existing work in the area of using (controlled) natural language as front-end for OWL and other related languages. In section 4.2, we bring an overview of related work, and in section 4.3, we look in detail into the work that is more relevant to ours, i.e. approaches that use a controlled English, and target OWL (or a similar language). Those approaches either propose authoring OWL ontologies in controlled English (we refer to it as CNL→OWL), or describe verbalization of OWL ontologies in controlled English (we refer to it as OWL→CNL). Only very few approaches propose a bidirectional interface to OWL, where the same controlled English is used for both authoring and verbalization.

4.2 Overview of related work

[Wil03] was one of the first papers to point to a need for natural language (English) interface to OWL. It describes an RDF verbalizer which is implemented as a pipeline of XSLT transformations, and points to an ongoing work on extending these transformations to DAML+OIL, the predecessor or OWL.

[HKKHW05] discusses the verbalization of OWL DL class descriptions. A part-of-speech tagger is being used to analyze the linguistic nature of class and property names and to split the names apart to form more readable sentences, e.g. the property restriction *ObjectHasValue(hasFlavor Strong)* is verbalized as "has Strong flavor". [HWGP$^+$06] extends this work to OWL individuals and property assertions. The authors also validate their approach by experimenting with seven university students, and find that the students significantly prefer natural language verbalizations to the DL-notation, and even more so to the functional notation, Turtle and RDF/XML.

[MS05a, MP07] discuss so-called "natural language directed inference" (which can be seen as a non-standard description logic reasoning task) to be applied to the ontology to make the verbalization of the ontology linguistically more acceptable, e.g. the (possibly complex) subsumers of a named class are presented in a way that they do not violate

the Grice's maxims.

[JKD06] discusses the verbalization of ORM ontologies on the basis of predefined templates (e.g. Mandatory, Exclusion, InterUniqueness). Each template provides natural language text for one of a set of supported languages.

[LW06] describes a verbalization of an XML-based rule interchange language R2ML[1] which (among other rule formats) can be translated into SWRL.

In a more mixed approach, the ontology editor COE[2] uses natural language labels (such as "isMotherOf must be at least 2") on otherwise graphical representation of ontologies.

The major shortcoming of all these approaches is that they lack any formal check that the resulting verbalizations are unambiguous. In this sense, a better approach is based on controlled natural languages which typically have a formal language semantics and come with a parser that could convert the verbalization back into the native OWL representation so that the verbalization is not a dead end, but rather a conversation turn in the machine-human communication.

[ST04] discusses a mapping between the controlled English PENG and various OWL subsets (RDFS, Description Logic Programs, etc.). [ST06] extends this work to cover OWL DL (without data properties) via a bidirectional mapping that is implemented as a Definite Clause Grammar. In this mapping, the *SubClassOf*-axiom is always written as an *if-then* sentence with explicit anaphoric references which for simpler axioms is unnecessarily hard to read. Largely on the basis of [ST04, ST06], a new syntax for OWL — Sydney OWL Syntax (SOS) — is developed in [CSM07].

[BCT07] provides a Categorial Grammar for a controlled English (Lite Natural Language) that expresses DL-Lite (a subset of OWL-Lite). Users may find Lite Natural Language somewhat unnatural since restrictions in DL-Lite are reflected in the syntax of Lite Natural Language, for instance that negations cannot occur on the left-hand side of the *SubClassOf*-axiom.

Rabbit [HDG07] is a fragment of English developed with the help of domain experts and intended to help writing OWL ontologies. The semantics of Rabbit includes controversial decisions, e.g. disjointness of named classes by default, *or* treated as "exclusive or", etc.

[FDT+07] describes a simple ontology authoring language CLOnE and its parser that is developed using the natural language processing framework GATE. The focus of this work is on generating OWL axioms from CLOnE sentences, but the OWL→CLOnE direction is also planned.

Note that all CNL-based approaches see natural language based authoring of OWL ontologies as a more important goal than mere verbalization of existing ontologies. I.e. they see their respective CNL-format as the main interchange format, and the traditional OWL syntaxes are treated more as machine code that is needed to be able to communicate with OWL reasoners.

[1] http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=R2ML

[2] http://cmap.ihmc.us/coe/

4.3 Detailed look on some related work

4.3.1 Sydney OWL Syntax

[CSM07] proposes a new syntax — Sydney OWL Syntax (SOS) — that can be used to write and read OWL ontologies in controlled natural language. Following Manchester OWL Syntax in making OWL more accessible for non-logicians, and building on the experience with Rolf Schwitter's PENG, the proposed Sydney OWL Syntax enables two-way translation and generation of grammatically correct English sentences to and from OWL 1.1 Functional-Style Syntax.

The work on SOS advocates a syntactically bidirectional mapping to OWL 1.1 Functional-Style Syntax. This means that e.g. *FunctionalObjectProperty* and a *SubClassOf*-axiom with the same meaning would be represented necessarily differently in SOS. For example,

(4.1) If X has Y as a father then Y is the only father of X. ($functional(hasFather)$)

(4.2) Everything has at most 1 father. ($\top \sqsubseteq \leq 1\ hasFather\ \top$)

At the time of writing, there is only a partial specification, and no implementation exists of a mapping of SOS to the standard OWL syntaxes.

4.3.2 Rabbit

Rabbit [DHK+07] (for a shorter overview see [HDG07]) is a fragment of English, used as an OWL-compatible knowledge representation language at Ordnance Survey[3] (Great Britain's national mapping agency). Rabbit is an integral part of the Ordnance Survey methodology [HDGK07, Goo07, HG07], which requires the domain expert and knowledge engineer to work together to produce ontologies — the domain expert is assumed to be familiar with Rabbit and is seen as the main author of Rabbit texts, the knowledge expert is assumed to be familiar with both Rabbit and OWL, and plays the main role in converting Rabbit texts into OWL.

Rabbit has been developed in close cooperation with domain experts and is mainly intended to help writing OWL ontologies, i.e. the focus is on the Rabbit→OWL direction. Examples of Rabbit sentences are (with the intended OWL representations in parentheses):

(4.3) A River is a concept. (*no corresponding OWL axiom*)

(4.4) A Duck Pond is a kind of Pond. ($DuckPond \sqsubseteq Pond$)

(4.5) A Father has at least 1 Child. ($Father \sqsubseteq \geq 1\, has\, Child$)

(4.6) A Father is uniquely defined as: is a kind of Male; has at least 1 Child. ($Father \doteq Male \sqcap \geq 1\, has\, Child$)

[3] `http://www.ordnancesurvey.co.uk`

These sentences illustrate some of the decisions made in Rabbit: all classes and properties must be declared by the domain expert (example 4.3); the difference between defined ('uniquely defined') and primitive classes is managed by the domain expert (example 4.6); sentences are kept short; class names are capitalized.

While the syntax of Rabbit is not too distinct from the syntax of ACE, its semantics differs in several points, and is in our opinion hard to (automatically) align with the semantics of OWL — disjointness of classes is assumed by default (and can be overridden with special Rabbit constructions), individuals have unique names, 'or' is treated as an exclusive OR, and the class description $\forall\, R\, C$ cannot be expressed unless it is conjoined with $\exists\, R\, C$, i.e. it is only possible to express $(\forall\, R\, C) \sqcap (\exists\, R\, C)$.

4.3.3 Lite Natural Language

[BCT07] provides a Categorial Grammar for a controlled English called Lite Natural Language (Lite NL) that expresses the description logic DL-Lite. DL-Lite [CGL$^+$05] is one of the tractable fragments of OWL 1.1, chosen in such a way that the algorithms for the main reasoning tasks have polynomial complexity. Nevertheless, DL-Lite is expressive enough to capture relational databases and UML diagrams.

From the natural language perspective, the restrictions in DL-Lite are not always easily explainable to an average user. For example, one can say in DL-Lite (and the corresponding Lite Natural Language):

(4.7) Every student who likes somebody is a man. ($student \sqcap \exists\, like\, \top \sqsubseteq man$)

(4.8) Every student does not like somebody. ($student \sqsubseteq \neg\, \exists\, like\, \top$)

but is not allowed to say

(4.9) Everybody who is not a woman is a man. ($\neg woman \sqsubseteq man$)

(4.10) Every student who likes a woman is a man. ($student \sqcap \exists\, like\, woman \sqsubseteq man$)

because negation and qualified existential quantification are not allowed on the left side of the *SubClassOf*-axiom due to restrictions in DL-Lite. Therefore, users may find Lite Natural Language somewhat unnatural and hard to learn because restrictions in DL-Lite are reflected in the syntax of Lite Natural Language, i.e. the grammar description of the language does not allow for simple rules such as "negation can be used everywhere".

Lite Natural Language is a subset of ACE, and also a subset of E2V if support for intransitive verbs and attributive adjectives were removed from Lite NL.

4.3.4 Jarrar et al

[JKD06] presents an approach to support multilingual verbalization of logical theories, axiomatizations, and other specifications such as business rules. This engineering solution is demonstrated with the Object Role Modeling (ORM) language [Hal01], although its underlying principles can be reused with other formal languages, such as description

logics, with the goal of improving the understandability and usability of the language by the domain expert.

They describe a flexible verbalization scheme based on templates (e.g. Subset, Subsumption, Exclusion) which can be ported to a new language in two hours. Supported are very different languages, in addition to English, e.g. German, Russian and Arabic. The verbalization schema simply handles each of the templates (i.e. formula types) separately, combining predicate names with predefined text to form sentences in the target natural language. Some examples of English verbalization are the following:

(4.11) Each Manager must be a type of Person. (Subsumption-template)

(4.12) Each Person must Has at least one BirthDate. (Mandatory-template)

(4.13) The combination of {BirthDate, Name} must refer to at most one Person. (InterUniqueness-template)

(4.14) Each Person WorksFor a Company must AffliatedWith that Company, and the other way around. (Equality-template)

(4.15) Each Manager who Manages a Company must WorksFor that Company. (Subset-template)

(4.16) Each Account OwnedBy Person or OwnedBy Company, or both. (ExMandatory-template)

(4.17) No Account can OwnedBy a Company and OwnedBy a Person. (Exclusion-template)

The verbalizations have been tested on 40 lawyers but the authors do not report how the tests were performed and what were the specific results.

As the approach is simple and general, the verbalization cannot be optimized for English. This is reflected in the above examples, where the morphological agreement is often violated and auxiliary verbs are dropped. It is also hard to understand which syntactic units the fixed phrases 'and the other way around' and 'or both' refer to.

4.3.5 CLOnE

[FDT+07] describes a simple ontology authoring language CLOnE (Controlled Language for Ontology Editing) and its conversion to OWL which is built on top of the GATE[4] framework. The grammar of CLOnE is composed of eleven sentence patterns which roughly correspond to eleven OWL axiom patterns. The following are examples of CLOnE sentences (their intended semantics is described in the parentheses).

(4.18) There are agents and documents. (declaration of named classes)

(4.19) Alice Jones and Bob Smith are persons. (typing of individuals by a named class)

[4] http://gate.ac.uk

(4.20) Universities and persons are types of agent. (*SubClassOf*-relation between named classes)

(4.21) Journals have articles. (declaration of an object property together with its domain and range)

(4.22) Projects have string names. (declaration of a data property together with its domain and range)

(4.23) SEKT has name with value 'Semantically-Enabled Knowledge Technology'. (data property assertion)

(4.24) Forget projects, journals and 'Department of Computer Science'. (instruction to the parser to undo certain declarations)

As seen from the examples, CLOnE targets only a tiny fragment of OWL, where one can express domain, range, subclass, and individual assertion axioms. Unnatural is the requirement that all names must be declared before they can be used. Sometimes the declarations are misleading, e.g. sentence 4.18 can be interpreted as: there actually exist agents and documents (i.e. individuals in the respective classes). This, however, is not the intended semantics of this sentence. Strange is also the mixing of imperative constructs into the otherwise declarative specification as in sentence 4.24.

Syntactic variation in CLOnE is provided by noun phrase conjunction which is equivalent to repeating the sentence for each of the nouns involved. Also, singular and plural nouns are not checked for syntactic agreement and are mapped to the same internal representation.

The usability of CLOnE has been tested on 15 volunteers with varying experience levels in ontology engineering. The subjects were asked to perform the same ontology engineering task with both Protégé and the environment incorporating CLOnE. The results of this study were favorable for CLOnE. It remains unclear, however, whether CLOnE can be extended (without major redesign) to express OWL constructs such as property restrictions and negation where the main problem with OWL's usability lies.

Chapter 5

ACE as a syntax for OWL

5.1 Introduction

On the one hand, the differences between ACE and OWL are huge. ACE — designed as a general knowledge representation language — is a subset of English, where English is given a precise and unambiguous semantics in first-order logic. The work on ACE has tried to find a compromise between (semantic) expressivity, usability, learnability and preciseness, while the decidability of the underlying logic has been secondary. OWL is a formal logic used to describe ontologies. Its design is guided by the need for reliable automatic reasoning (i.e. it is algorithmically decidable and scalable) and the need for certain expressivity for naturally occurring knowledge engineering tasks. Although usability and learnability have been listed as part of the goals in the OWL specification [Hef04] (see "Ease of use"), their satisfactory implementation has so far been secondary.

On the other hand, both ACE and OWL are meant for knowledge representation, and as such, have similar goals. Interoperability between those languages brings several benefits to the users of both languages. ACE users would gain an access to existing ontologies and reasoners. OWL users would gain an alternative user-friendly syntax and a different view on ontology engineering.

This chapter describes a bidirectional mapping between ACE and OWL (which we refer to as ACE↔OWL). In section 5.2, we discuss the main differences between ACE and OWL that make the mapping non-trivial; in section 5.3, we discuss the main design decisions that have guided us in the developing of the mapping; in section 5.4, we discuss how OWL class, property, and individual names are matched with the English content words used in ACE; in section 5.5, we specify the translation of ACE texts into OWL ontologies; in section 5.6, we talk about the "other direction", i.e. verbalizing OWL ontologies in ACE; and finally in section 5.7, we point out some caveats of our approach.

5.2 Differences between ACE and OWL

The following is a list of some concrete differences between ACE and OWL.

- The usage of words in ACE is restricted by English rules and conventions. One has to respect the English syntax and morphology in order to write understandable (and acceptable) ACE sentences. In OWL, names (i.e. names of classes, properties and individuals) can be almost arbitrary sequences of characters. Some orthographic guidelines are sometimes given (e.g. class names should start with a capital letter), but the user is not really under any restrictions (e.g. the OWL editors do not currently enforce any naming style).

- OWL class descriptions can be made arbitrarily complex by using parentheses. English, on the other hand, does not use parentheses for grouping. Similarly, ACE relies on a few predefined scoping and binding rules to achieve some forms of grouping, but cannot handle scopes of arbitrary complexity.

- OWL has a set of non-structural restrictions, i.e. rules which remove some (otherwise) syntactically legal constructions from the language. (E.g. properties which have been declared as transitive cannot be used in cardinality restrictions.) Those rules achieve that certain reasoning tasks on OWL ontologies remain decidable. ACE does not have such restrictions.

- OWL (especially OWL 1.1) defines many short-hand constructs (i.e. constructs the semantics of which can be achieved by a set of more basic constructs, but which are needed as people often use them). In controlled natural languages, such, essentially macro facilities, are harder to provide.

- OWL supports meta-statements so that users can e.g. label and annotate axioms and other constructs, express version information, import other ontologies, and deprecate some statements, all in the same ontology language. ACE does not support this language-internally but one can of course add comments to parts of the text, place ACE texts under version control, concatenate several ACE text into one, "comment out" certain sentences etc.

5.3 Main design decisions

5.3.1 Introduction

In this section, we discuss the main design decisions for the ACE-fragment that is to be used both as an OWL-compatible ontology language and as an English-based verbalization language for OWL ontologies.

5.3.2 Reversibility

We first discuss the desired properties of the translations ACE→OWL and OWL→ACE, specifically, the "round-trip translations" ACE→OWL→ACE and OWL→ACE→OWL, and how they preserve the meaning expressed in the original language (either ACE or OWL). There are several constrains that could be placed on the translations to assure

that they are meaning preserving. We follow the discussion in [Sow05] that lists the following constraints: invertible, truth preserving, vocabulary preserving, and structure preserving.

Invertible

Invertibility here means that a mapping f from language L_1 to language L_2 is defined for every sentence s of language L_1 and has an inverse function g that maps sentences in language L_2 back to sentences in language L_1.

As both ACE and OWL have various incompatibilities, we cannot achieve mappings that are defined on all ACE sentences or all OWL axioms. Instead, we look for a subset of ACE, ACE_1 each of whose sentences can be mapped to a subset of OWL, OWL_1. Also, we look for a subset of OWL, OWL_2 each of whose axioms can be mapped to a subset of ACE, ACE_2. As the subsets ACE_1 and ACE_2 (OWL_1 and OWL_2) are not necessarily equivalent, it is possible that a mapping is not invertible. In our design of the respective subsets, however, we make sure that this does not happen often and does not cause any end-user confusion. We design ACE_1 as the largest fragment of ACE which can be translated into OWL. We design ACE_2 as a fragment of ACE which is best suitable for verbalizing OWL ontologies.

Preserving vocabulary

Vocabulary preserving translations must keep all the words (i.e. (lemmas of) ACE content words) and names (i.e. names of OWL classes, properties and individuals) unchanged in the end of the respective round-trip. Furthermore, no new words/names must be generated.

This is an important requirement. The names of classes, properties and individuals are visible to the user in all OWL editors (while the exact shape of axioms is not, in many cases). In all those tools, the user (and only the user) is in control of introducing new names or changing/removing existing names. It would be confusing if a round-trip translation through an ACE representation changed the names. Similarly, and for the same reasons, all ACE content words must be preserved and no new words should be introduced in the round-trip.

We note that in some cases it might be desirable to introduce new names, e.g. it would simplify the syntax of axioms/sentences and sometimes make them more readable if all occurrences of a complement of a named class were replaced by a new name (by $\neg C \mapsto nonC$ which would be supported by an additional definition $nonC \doteq \neg C$). Nevertheless, we think that the user should be in control of such transformations.

Preserving structure

Preserving structure would mean that every OWL element (e.g. *SubClassOf*, *FunctionalObjectProperty*, etc.) is preserved in OWL→ACE→OWL. We consider this property not important because the OWL axioms are hidden from the users in most end-user tools.

Still, it might be desirable to keep the number of axioms the same in the mapping (and e.g. not split an axiom into several axioms) because users might be observing a metric like "number of statements in the knowledge base". Note that we violate the principle of preserving the axiom-count when verbalizing *DisjointUnion* and other similar shorthand axioms (see section 5.7.4).

In the other direction, preserving structure would mean that every ACE sentence is preserved in ACE→OWL→ACE. Again, this property is not so important as such a structure-changing translation can be considered "paraphrasing" or "normalization" which can be seen as something useful (i.e. a feature).

Preserving truth

Preserving truth means that the round-trip produces a sentence or an axiom which is logically equivalent to the original. Fulfillment of this requirement is obviously important and is the core design decision.

A truth-preserving mapping can start off by applying truth-preserving transformations within the language itself. For example, in the OWL→ACE direction, OWL's *FunctionalObjectProperty* can be first expressed via *ObjectMaxCardinality*, or in the ACE→OWL direction, the DRSs corresponding to ACE sentences can be manipulated to remove e.g. double negations. As far as preserving truth is concerned, different ACE sentences can map to the same OWL axiom, or different OWL axioms can map to the same ACE sentence. I.e. the mappings do not have to be necessarily structure preserving. We will exploit this property to deal with OWL's syntactic sugar, most of which does not have a direct natural counterpart in ACE nor English.

In summary, we want to select an ACE fragment as large as possible which can target an OWL fragment as large as possible, so that the bidirectional mapping between those fragments preserves vocabulary and truth, but does not necessarily preserve the structure of the input sentences or axioms, although, some forms of structure preserving are desirable and also attempted.

5.3.3 Compatibility with ACE semantics

We can imagine a mapping between OWL and a subset of ACE, such that the mapping preserves the meaning of OWL axioms in the round-trip OWL→ACE→OWL but violates the semantics of ACE. For example, if we map the OWL axiom $C \sqsubseteq D$ to the ACE sentence *A C is a kind of a D.* and this sentence cannot be generated from any other OWL axiom that is not logically equivalent to $C \sqsubseteq D$, then the mapping is reversible. Still, the fact that the FOL representation of the OWL axiom contains a universally quantified variable is not preserved in the ACE representation because the indefinite article 'a' introduces always an existential quantifier for the corresponding discourse referent. Therefore, this mapping is not compatible with the semantics of ACE. The OWL axiom

must instead be mapped to a universally quantified (i.e. *if-then* or *every*) sentence, e.g. *Every C is a D.*. Also, care must be taken, so that the ACE scoping rules and binding order as discussed in section 2.4 are respected.

5.3.4 Acceptable and understandable English

The verbalization (i.e. the mapping from OWL to ACE) must result in an understandable and acceptable English. We consider the fulfillment of this requirement guaranteed by the ACE design decisions. Furthermore, in the verbalization direction, we use ACE constructs, such as relative clauses, which provide conciseness, and in order to increase readability, we also try to avoid anaphoric references. For instance, we prefer the *every*-sentence "Every man is a human." to the (in ACE) semantically equivalent *if-then* sentence "If there is a man then he is a human.".

No logic-specific words ('object property', 'functional', 'transitive', etc.), and no logic-specific symbols (e.g. parentheses, $\exists$, $\forall$, $\sqsubseteq$) must be used in the verbalization, because such words/symbols are specific to the domain of logic and are not common in everyday language. While ACE does not include symbols like $\exists$ or $\sqsubseteq$, it does include content words like 'class', 'functional' and 'property'. It is therefore possible to write

(5.1) Human is a class.

(5.2) Identify is a property that is functional.

However, these words carry no predefined meaning in these sentences, as then they would essentially be function words, and this would violate the semantics of ACE which only allows for a closed class of predefined function words.

In the nutshell, the verbalization of OWL ontologies must be everyday English, apart from (possibly domain specific) content words but those words must obey English morphology and syntax.

5.3.5 Ontology as plain text

While the verbalization would profit from syntax-aware highlighting and layout, cross-linking, etc., we set out to produce just plain text. The advantage of plain text is that it can be used in text-only environments. For example, the ontology can be described audibly (through the headphones) to the driver of a car or airplane.

5.3.6 Independence from tools

The designed language must not depend on any tool for reading and writing the sentences. There are many types of tools that can support the production of controlled natural language texts, ranging from tools that provide syntax highlighting, automatic indentation, and auto-completion of words, to tools that integrate a predictive editor (see e.g. ECOLE [SLH03] and AceWiki [FKK07e]), paraphraser, or a reasoner for syntactic and semantic feedback. Studying and designing such tools is a large research topic on

its own and outside the scope of this thesis. Here we design a language that can be used in a tool-independent way, e.g. the user should be able to use his/her favorite text editor and not be forced to learn to use a new one.

5.4 Names

5.4.1 Introduction

The atomic units in ACE sentences are words and fixed phrases — function words like e.g. 'every', '10', 'does not', and content words like e.g. 'man', 'own', 'John'. The atomic units of the OWL syntax are class, property and individual names, which are connected to each other by various logical operators/functions, e.g. *SubClassOf*, *ObjectSomeValuesFrom*.

While OWL makes a distinction between classes, properties and individuals, natural language does not explicitly make such a distinction, e.g. there is no clear-cut morphological category that would denote that a word is either a class, property or individual. Still, an easy-to-follow mapping between ACE word-classes and OWL names is attainable due to a number of similarities. E.g. the proper names in ACE are interpreted as denoting an object of the domain which in its first-order logic representation is always existentially quantified. Similarly, OWL individuals are interpreted as constants in first-order logic. That is, proper names can be mapped to individuals and vice versa. ACE transitive verbs take a nominal subject and object. Similarly OWL object properties are always connected to two classes or individuals. Because in the FOL representation, the OWL properties are binary predicates, ACE transitive verbs can be mapped to OWL properties, and vice versa. Because in the FOL representation, OWL named classes are unary predicates, ACE common nouns can be mapped to OWL named classes, and vice versa.

Looking at various OWL ontologies, most of which use English words or word combinations for individual, class, and property names, we see that the choice for names for individuals, classes and properties more or less matches the word-classes proper name, noun, and (transitive) verb.

In the following, we motivate the decision of using common countable nouns, transitive verbs and proper names to express OWL classes, properties and individuals. Special discussion is given to passive verbs (as inverse properties). We also discuss punning (i.e. the fact that OWL 1.1 allows individual names to be used as class names, etc.). Similarly, ACE does not require that the word-classes of common nouns, verbs and proper names are disjoint, as this overlap is quite common in English.

5.4.2 Internationalized Resource Identifiers

OWL ontologies and their named elements are identified using Internationalized Resource Identifiers (IRIs) that are defined in RFC-3987[1]. IRI is defined by extending the

[1] `http://www.ietf.org/rfc/rfc3987.txt`

syntax of URIs to a much wider repertoire of characters (Unicode instead of US-ASCII).

As IRIs are global identifiers, they tend to be long, and thus various abbreviation mechanisms are supported by OWL syntaxes, e.g. XML-based syntaxes can use XML entities or the *xml:base*-attribute[2] to define the invariable part of the names just once, and then use a combination of an XML entity and the variable part of a name through-out the ontology to attain a readable form of writing.

ACE could achieve the same effect by introducing a notion of namespace.[3] This involves separating the words into the local name and the namespace identifier part, e.g. in

[http://google.com/employee#]John

'John' is the local name, and 'http://google.com/employee#' is the namespace identifier. To be able to abbreviate namespace identifiers, a form of meta-statement is needed, e.g.

yahoo := [http://yahoo.com/employee#].

where 'yahoo' is declared to be an abbreviation of 'http://yahoo.com/employee#'. The resulting abbreviation would be an alternative (much shorter) way to qualify the names. It could either use the same bracket-syntax as full namespace identifiers (i.e. '[yahoo]John') or a slightly shorter (and more standard) one with a colon-sign, i.e. 'yahoo:John'. An empty abbreviation would naturally mean that the namespace is the default, i.e. it applies to all words that are not explicitly qualified by some other namespace.

The following is an example of an ACE text that makes use of such namespace declarations.

```
# Every word belongs to the namespace 'http://attempto.ifi.uzh.ch/names/'
:= [http://attempto.ifi.uzh.ch/names/].

# We can write ACE sentences as usual,
# without explicitly pointing to the namespace.
John likes Mary.
Everybody likes Mary.

# We can use words from different namespaces in the same sentence.
# The ''local'' word 'John'
# is identical to 'http://microsoft.com/employee#John' but
# is not identical to 'http://google.com/employee#John'.
John is [http://microsoft.com/employee#]John
 and is not [http://google.com/employee#]John.

# Namespace identifiers can be given abbreviations ...
fee   := [http://feelings.net/].
foo   := [http://food.net/].
```

[2] http://www.w3.org/TR/xmlbase/

[3] At the time of writing, support for expressing IRIs/namespaces is not implemented in ACE. Thus, the following description is a proposal of how to match OWLs support for IRIs in ACE.

```
yahoo := [http://yahoo.com/employee#].

# which can later be used to write shorter sentences.
John fee:likes yahoo:Mary and foo:likes an [foo]ice-cream.
```

Note that such qualifying is only possible with ACE content words and not with function words.

The DRS-conditions that correspond to the ACE content words would use 2-place functions in the second argument position in order to specify the name and its namespace.

(5.3)

A B C
object(A, name(John, http://attempto.ifi.uzh.ch/names/), named, na, eq, 1) object(B, name(Mary, http://yahoo.com/employee#), named, na, eq, 1) predicate(C, name(like, http://feelings.net/), A, B)

In the following, we make the simplifying assumption that all names come from the same namespace which therefore does not have to be made explicit, i.e. we do not discuss namespaces anymore.

5.4.3 Common nouns as named classes

In OWL ontologies, class names are usually nouns or nouns preceded by an attributive adjective. For example, [MS05b] analyzes the linguistic nature of class and property names in 882 public OWL ontologies and find that class names predominantly fall into the category of noun or noun phrase. They analyze 37,260 different class names and find that 72% of the class names end with an English noun (that exists in WordNet). Also, 30% consist entirely of strings of nouns.

It thus seems reasonable to choose the ACE counterpart of OWL class names to be nouns as well. We only have to decide on a few more detailed cases. Are both countable and mass nouns allowed? Are both singular and plural forms (e.g. ‘man’, ‘men’) allowed? Are both lowercase and uppercase (e.g. ‘man’, ‘Man’) allowed? Are multi-word units (e.g. ‘bus driver’) allowed?

In the following, we consider only countable common nouns (either in lowercase or uppercase) which at the level of the DRS are represented by the condition-pattern *object(_, Name, countable, na, _, _)*, where *Name* is the noun from the ACE text in lemmatized form. Note that ACE does not support real multi-word units, i.e. all such units must be hyphenated together and thus would cause no syntactic complications.

Note that ACE offers also other word-classes which have similarities to nouns in their syntactic behavior, e.g. predicative adjectives (“a man is *rich*”) and intransitive verbs (“a man *waits*”). Extending the mapping to these word-classes is trivial, but at the same time would give up the requirement of an injective mapping to OWL, e.g. the noun ‘male’ and the adjective ‘male’ would both be mapped to a class name ‘male’, i.e. their distinction would be lost at the OWL level unless some meta information is included to disambiguate between the original word-classes.

5.4.4 Proper names and top-level common nouns as individuals

Proper names play well the role of OWL individuals as in natural language they stand for an individual, e.g. one cannot quantify (over) a proper name. At the DRS level, a proper name is represented by a top-level object-condition with the pattern *object(_, Name, named, na, eq, 1)*, where *Name* is the proper name from the ACE text. Note that we only consider singular proper names.

OWL also supports anonymous individuals, i.e. individuals that are not assigned a global IRI, because it is unknown or unimportant. Such individuals can nevertheless be used in OWL expressions, e.g. in order to assert that "there is **an animal** that every man likes and that every woman hates". As a counterpart, in ACE, one can write e.g. "There is an animal." to get a top-level (existentially quantified, and not in the scope of universal quantification) DRS object-condition. An object introduced in this way can be seen as an individual without a name but which can nevertheless be anaphorically referred to later in the text (e.g. "Mary likes the animal."). Such nouns have the same DRS condition-pattern as regular nouns but in this case we restrict them to be singular, i.e. *object(_, Name, countable, na, eq, 1)*. Note that top-level indefinite pronouns 'somebody' and 'something' are also treated as anonymous individuals, even though their object-conditions are slightly different, namely *object(_, somebody, countable, na, eq, 1)* and *object(_, something, dom, na, na, na)*, respectively.

5.4.5 Transitive verbs as object properties

In existing OWL ontologies, property names are more diverse than class names in terms of their linguistic word-classes. [MS05b] analyzes 1354 different property names and find that 42% of the property names end with an English verb. However, there are also many patterns which are based on nouns — *Noun*, *Verb + Noun*, *Noun + Preposition*, and *Verb + Verb + Noun* constitute 16% of the patterns. Unfortunately, their study does not discuss object properties and data properties separately. (It seems that data properties are often nouns, and it would be thus interesting to describe them by separate statistics.) They also do not discuss how much passive verbs are used in property names.

ACE provides several word-classes and constructions which are verb-like in the sense that they take nominal arguments. In addition to transitive verbs (e.g. 'like', 'smile-at'), one can use transitive adjectives (e.g. 'fond-of'), comparative adjectives (e.g. 'taller than', 'as tall as'), and *of*-constructions and Saxon genitives (e.g. 'father of', 'John's father').

In the following, we focus only on transitive verbs, both in singular 3rd person ('likes') and infinitive ('does not like', 'do like') forms. They have the condition-pattern *predicate(_, Name, _, _)*, where *Name* is the lemma of the transitive verb from the ACE text. Again, extending the mapping to object-requiring adjectives is trivial but would hamper the injective nature of the mapping. As *of*-constructions seem to naturally occur in ontologies (e.g. in property names like *fatherOf*), we discuss them as a special case in section 6.2.

Passive verbs as inverse object properties

OWL allows inverting the properties without giving a name to the new property, i.e. by the *InverseObjectProperty*-construct. While the syntax of OWL is rather fixed, ACE allows more flexibility and inverse properties can be expressed either by exchanging the subject and the object arguments of a transitive verb, using object (object-extracted) relative clauses, or using passive sentences. Passive verbs seem to offer the most flexible solution by using a dedicated morphological form, namely the past participle. Furthermore, users often "think in passive" when writing Semantic Web ontologies, this is reflected in the names given to object properties. I.e. a user is equally likely to state:

(5.4) Every author is somebody who writes a book.

(5.5) Every book is something that is written by an author.

ACE provides two ways to avoid the passive construction: using object relative clauses, or anaphoric references. E.g. sentence 5.5 can be reformulated by

(5.6) Every book is something that an author writes.

(5.7) If there is a book then an author writes the book.

Still, these constructs seem less natural than the passive. Also, they can be problematic when it comes to readability, especially with more complex descriptions. Consider for example

(5.8) Every book is something that an author that a publisher that John hates knows writes.

which heavily uses center embedding which is well-known to cause difficulties (i.e. memory load) in understanding sentences (see e.g. [Gib98, Lew96]). The Core ACE equivalent

(5.9) If there is a book X1 then the book X1 is something X2 and John hates a publisher X3 and the publisher X3 knows an author X4 and the author X4 writes X2.

uses many anaphoric references, also making the reading difficult. With passive support, however, the previous sentences could be expressed as

(5.10) Every book is something that is written by an author that is known by a publisher that is hated by John.

We finally note that the DRS representation of passive verbs is the same as for regular transitive verbs, with the only difference that the subject and object arguments of the respective predicate-condition are exchanged.

5.4.6 Overlap between word-classes

We have to be concerned about the possible overlap between word-classes, i.e. the fact that the lexicon might list the same word form as belonging to many categories (e.g. a word is both a verb and a noun, such as 'run'). This, however, poses no problem for parsing as syntactically the word-classes are clearly separated. Even common nouns and proper names which are similar in that they both occur as arguments of verbs, can be separated on the basis of either always requiring a preceding determiner (such are common nouns) or never allowing a determiner (such are proper names), i.e. in all cases the word-class of a word can be reliably guessed on the basis of its context.

Still, there are a few cases that we need to pay attention to. We need to require no overlap between function words and content words, otherwise, in e.g.

(5.11) Every recursive-function calls itself.

the word 'itself' can be interpreted as either a reflexive pronoun or a (lowercase) proper name (as it is preceded by no determiner). Similarly, variables which are used to make anaphoric references to nouns, require no determiners and thus overlap with proper names in their syntactic behavior.

Note that all such cases are already resolved and explained by ACE interpretation rules. E.g. something that can syntactically be a proper name must be looked up in the lexicon (by the ACE parser), and if it is missing then it is considered to be a variable.

Tolerating an overlap between content word-classes fits well with OWL's idea of punning. Punning allows the same IRI to be simultaneously used as an individual name, class name, and property name. Its function is resolved by the context. Thus ACE's behavior is compatible with OWL's in this case.

5.4.7 Reversibility of morphological mappings

To avoid problems with ACE words (OWL names) changing as a result of the round-trip, we require the lexicon to be bijective, i.e. for each lemma in some word-class there is exactly one surface form of the same word-class, and vice versa. This means e.g. that a plural noun like 'leaves' which in English could be lemmatized either as 'leaf' or 'leave', in our subset of ACE would have a unique lemmatization. This avoids the following "distortion" of the class description $(\geq 3\ \mathit{has\ leaf})$ in following the round-trip.

(5.12) $\geq 3\ \mathit{has\ leaf} \longmapsto \ldots$ that has at least 3 leaves $\longmapsto \geq 3\ \mathit{has\ leave}$

Fortunately, the number of such words in English is very small, in addition to 'leaves', there are e.g. axis/axe $\mapsto$ axes, basis/base $\mapsto$ bases, ellipsis/ellipse $\mapsto$ ellipses. In the other cases, the clashing words are synonymous or spelling variants (e.g. junkie/junky $\mapsto$ junkies).

In the other direction, a singular noun (i.e. lemma) like 'hoof' which in English has two possible plural forms 'hoofs' and 'hooves', in our subset of ACE has just one plural. This avoids the following distortion.

(5.13) ... that has at least 3 hoofs $\longmapsto \geq 3$ *has hoof*
$\longmapsto$ ... that has at least 3 hooves

Again, the number of such words in English is small and has to do with spelling variation and not change in word sense (e.g. 'platypi'/'platypuses', 'elk'/'elks', 'dominos'/'dominoes').

The latter case where a lemma can have several surface forms also occurs with the mapping of infinite verbs to past participles, e.g. 'shaved' and 'shaven' are both acceptable English forms of 'shave'. Again, as with nouns, this spelling variation does not create a change in word sense, and one of the variants can be dropped.

5.5 Translating ACE into OWL

5.5.1 Introduction

In this section, we describe a method of translating ACE texts into OWL ontologies. The ACE→OWL translator accepts as input the language described in section 2.3, with the exception of interrogative sentences, genitives (*of*-constructions and their semantic equivalents), and numbers and strings as noun phrases. Those exceptions will be covered as part of the extensions in section 6. In addition to those constructs, some additional structures are rejected by the translator due to an "incompatible" use of anaphoric references. In the following, we will call the accepted ACE subset ACE_1, see section 5.5.2.

We rely on the existing solution of mapping ACE texts into DRSs, in order to be able to work on the more convenient DRS level where some of ACE's syntactically different but semantically equivalent sentences are represented in the same form. We first rewrite the DRS in order to remove all implication-conditions that are contained in embedded DRSs (see section 5.5.3). The resulting DRS is translated by a "rolling up" technique into a set of OWL axioms (see section 5.5.4). The algorithm (described in section 5.5.6) either finds the OWL axioms that correspond to the DRS, or rejects the DRS with a (user-level) error message. These error messages are described in section 5.5.7.

The translation method is correct in the sense that all translation steps preserve the semantics of the original input. The translation method is not complete in the sense that there exist certain ACE sentences which while having a natural representation in OWL, are rejected by the algorithm. Also, some forms of OWL axioms cannot be expressed in ACE due to their complex structure. We describe these two forms of incompleteness in section 5.5.8.

We also note that (similarly to the official OWL syntaxes) ACE allows one to write ontologies which are not in OWL because they violate the non-structural restrictions of OWL. The ACE→OWL mapping does not capture such sentences. This is left to OWL reasoners.

5.5.2 ACE_1 construction rules

ACE_1 is a subset of the ACE fragment that was described in section 2.3, and is defined by the following restrictions on this fragment:

- interrogative sentences are not allowed;
- *of*-constructions, Saxon genitive, possessive pronouns, and relative clause pronoun 'whose' are not allowed;
- numbers and strings as noun phrases are not allowed;
- copula complements cannot be plural nouns ('John is 3 men.').

In addition to those constructs, some additional structures are rejected by the translator due to an "incompatible" use of anaphoric references (5.5.7).

5.5.3 Removing embedded implications

The DRS→OWL translation algorithm works on a simplified DRS language where the *ReferentList*-argument of the DRS is stripped from the top-level DRS and from all the embedded DRSs, resulting in a condition list with possibly embedded condition lists.

$$drs(ReferentList, ConditionList) \longmapsto ConditionList \tag{5.14}$$

This transformation does not lose any information because discourse referents are globally unique.

As the first step of converting condition lists into OWL axioms, we modify some conditions so that they are easier to handle by the subsequent processing steps. Namely, we rewrite all embedded implication boxes (i.e. those implications that are not in the top-level DRS but are embedded in disjunctions, negations or top-level implications). We also remove double negations.

$$CL_1 \Rightarrow [CL_2 \Rightarrow CL_3] \longmapsto (CL_1 \oplus CL_2) \Rightarrow CL_3 \tag{5.15}$$

$$CL_1 \Rightarrow CL_2 \longmapsto \neg(CL_1 \oplus [\neg CL_2]) \quad \text{if } CL_1 \Rightarrow CL_2 \text{ is embedded} \tag{5.16}$$

$$\neg[\neg CL] \longmapsto CL \tag{5.17}$$

where CL, CL_1, CL_2, CL_3 are condition lists, $[C]$ is a condition list containing a single condition C, and $\oplus$ is a merging operator that simply concatenates two condition lists.

This rewriting lets us handle ACE sentences that are parsed into a DRS where implication-conditions occur inside embedded DRSs. E.g. the following sentences

(5.18) If there is a goat then everything that the goat eats is an apple.

(5.19) It is false that every student is a freshman.

(5.20) John does not like every dog.

(5.21) For every thing X for every thing Y if X owns something that contains Y then X owns Y.

would be seen as equivalent to the sentences

(5.22) If there is a goat then it is false that the goat eats something that is not an apple. (= No goat eats something that is not an apple.)

(5.23) There is a student that is not a freshman.

(5.24) There is a dog that John does not like.

(5.25) If a thing X owns something that contains a thing Y then X owns Y.

that either do not have any implications or all the implication-conditions occur directly in the top-level DRS. In other words, we transform some logically equivalent DRSs into lexically equivalent forms.

5.5.4 Rolling up the condition lists

Our approach to translating condition lists to OWL axioms uses the "rolling up" technique which is also used in [HT02] (see [Tes01] for the details, and [Gli04] for an easily readable overview) for conjunctive query answering, and in [PSG+05] to translate a subset of SWRL to OWL DL. In this approach, a conjunction of unary and binary predicates, containing either variables or constants (individuals) is transformed into description logic class descriptions. In order to do that, the conjunction is seen as a graph. For example,

(5.26) $$node1(x) \wedge edge1(x,y) \wedge edge2(y,z) \wedge edge3(x,w) \wedge node2(w)$$

corresponds to the graph with 4 nodes (x, y, z, w), where the node x is labeled 'node1' and connected to nodes y and w via edges labeled 'edge1' and 'edge3'. The node y is connected to node z via an edge labeled 'edge2'. The node w is labeled 'node2'. Because this graph has no cycles and confluent nodes (where more than one edge enters a node that corresponds to a variable, e.g. $P(x,z) \wedge Q(y,z)$), it can be represented by a pair of a distinguished variable (the root of the graph) and a description logic class description, in our case

(5.27) $$(x,\ node1 \sqcap \exists\, edge1\ (\exists\, edge2\ \top) \sqcap \exists\, edge3\ node2)$$

where conjunction in 5.26 is transformed either into an intersection or property restriction.

In our approach, we treat the discourse referents of named and top-level object-conditions as constants, all other object-conditions as unary predicates, and predicate-conditions as binary predicates (note that we only support those predicate-conditions that

have been derived from transitive verbs). Each complex condition (implication boxes, disjunction boxes and the negation box) is rolled up independently into a pair of a distinguished variable and a description logic class description, such that the following constraints are respected.

- Every condition list corresponds to a directed graph (having exactly one root node, the distinguished node). The underlying undirected graph can only contain cycles of length one (to be able to handle *ObjectSelfRestriction*).
- Every pair of implication-boxes shares their root nodes.
- Every pair of disjunction-boxes shares their root nodes.
- Every pair of disjunction-boxes shares their root node with exactly one node in a box that is one level higher.
- Every negation-box shares its root node with exactly one node in a box that is one level higher.

The algorithm maps object-conditions either to named classes or to *ObjectOneOf*-descriptions that contain just one individual. Predicate-conditions are mapped to *ObjectSomeValuesFrom*-descriptions or general cardinality restrictions. Depending on the order of the arguments of the predicate-condition, the object property can either be explicitly named or inverted (i.e. *InverseObjectProperty*). Such inverting gives us more flexibility in the process of rolling up, i.e. sometimes we can avoid confluent nodes. The disjunction boxes translate to *ObjectUnionOf*-descriptions, the negation boxes to *ObjectComplementOf*-descriptions, and the co-occurrence of conditions in the condition list maps either to *ObjectIntersectionOf*, or is treated by an embedding into property restrictions. The implication boxes translate to OWL *SubClassOf*-axioms.

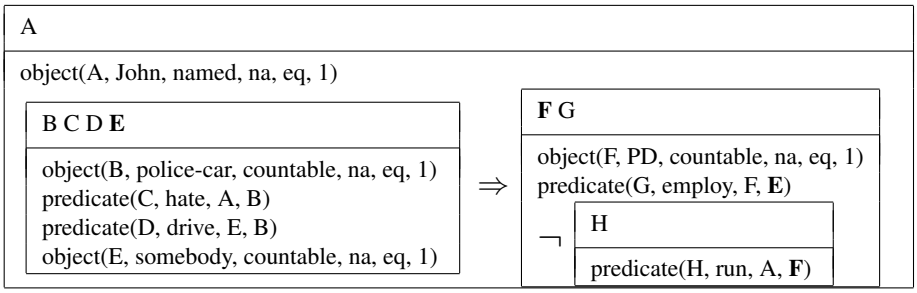
5.5.5 Example

Before we describe the DRS→OWL algorithm formally, let us look at an example. Given an ACE sentence

(5.28) Everybody who drives a police-car that John hates is employed by a PD that John does not run.

the ACE parser translates it to the DRS

(5.29)



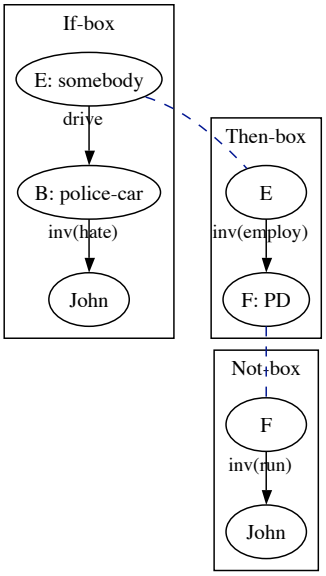


Figure 5.1: A directed graph corresponding to the DRS of the ACE sentence "Everybody who drives a police-car that John hates is employed by a PD that John does not run.". The inverted predicates *hate*, *employ*, and *run* are represented by the function $inv(\cdot)$. Node sharing is represented by dashed lines.

As the DRS contains no double-negations or implication-conditions that are contained inside embedded boxes, we can skip the DRS rewriting step. As the *if*-box and the *then*-box share exactly one argument ('E'), and the *then*-box and the negated box share exactly one argument ('F'), the structure can be seen as a tree structure (figure 5.1) which in turn translates to the OWL *SubClassOf*-axiom

$$\top \sqcap \exists\, drive\, (police\text{-}car \sqcap \exists\, hate^{-}\, \{John\}) \sqsubseteq \exists\, employ^{-} (PD \sqcap \neg(\exists\, run^{-} \{John\})) \tag{5.30}$$

5.5.6 DRS→OWL algorithm

The DRS→OWL algorithm consists of three parts to handle two different types of implications and the top-level condition list.

1. Handling the implication boxes that map to *SubClassOf*-axioms.
2. Handling the implication boxes that map to *SubObjectPropertyOf* and *DisjointObjectProperties* axioms (if mapping to *SubClassOf* fails)

3. Handling the top-level box with embedded negations and disjunctions.

We describe the algorithm declaratively, as a set of Prolog rules. The DRS conditions are Prolog terms where the complex conditions (implication-boxes, disjunction-boxes and the negation box) are represented by terms `=>/2`, `v/2`, and `-/1` that take (condition) lists as arguments. In the code, they are written as

- `[ ... ] => [ ... ]`
- `[ ... ] v [ ... ]`
- `- [ ... ]`

The core rules work on condition lists that are expressed as Prolog lists. The rules instantiate a list of Prolog terms that correspond to the OWL axioms or descriptions in Functional-Style Syntax. In addition to those two lists, the two further arguments to most rules are the distinguished discourse referent (either instantiated or not) and information about the top-level discourse referents. Note that most rules make use of the Prolog's built-in `select(?Elem, ?List, ?Rest)` to select an element `Elem` from a list `List` and access the remaining part `Rest` of the list.

Data structures

The algorithm makes use of a data structure `RefList` which is a list of `ref_oneof/2`-terms where each term maps a discourse referent to its corresponding OWL *ObjectOneOf*-class. We generate this data structure by applying `findall/3` on the list of top-level conditions as

```
findall(
        ref_oneof(X, OneOf),
        (
                member(Condition, Drs),
                condition_oneof(Condition, X, OneOf)
        ),
        RefList
)
```

where `condition_oneof/3` is defined by

```
%% condition_oneof(+Condition:term, -Ref:term, -ObjectOneOf:term)
%
% @param Condition is a DRS (top-level) condition
% @param Ref is a discourse referent
% @param ObjectOneOf is an ObjectOneOf-class corresponding to the condition
%
condition_oneof(object(X, Name, named, _, _, _)-_, X,
                                        'ObjectOneOf'(['Individual'(Name)])).
condition_oneof(object(X, _, countable, _, _, _)-_, X,
                                    'ObjectOneOf'(['AnonymousIndividual'(X)])).
condition_oneof(object(X, something, dom, _, _, _)-_, X,
                                    'ObjectOneOf'(['AnonymousIndividual'(X)])).
```

i.e. named-object-conditions (which stand for proper names) map to OWL named individuals wrapped in the *ObjectOneOf*-class, and other top-level object-conditions map to anonymous individuals, again, wrapped in the *ObjectOneOf*-class. In the case of anonymous individuals, a non-standard function *AnonymousIndividual* that identifies the individual by the name of the discourse referent is returned. This makes it possible to differentiate between named and anonymous individuals.

Common rules

`is_toplevel/3` takes the discourse referent and a list of top-level discourse referents as arguments and if the given referent is among the top-level referents then returns an OWL individual (wrapped into *ObjectOneOf* to make a class).

```
%% is_toplevel(+Ref:nvar, +RefList:list, -ObjectOneOf:term)
%
% @param Ref is a discourse referent
% @param RefList is a list of top-level discourse referents
% @param ObjectOneOf is an ObjectOneOf-class
%
is_toplevel(Ref, RefList, ObjectOneOf) :-
        member(ref_oneof(Ref, ObjectOneOf), RefList).
```

`get_namedclass/2` maps ACE nouns and indefinite pronouns to named classes. Note that 'a thing' is equivalent to 'something' but it is recommended to use it only in the plural constructions, e.g. 'at least 3 things' where ACE does not allow one to say 'at least 3 somethings'.

```
%% get_namedclass(+Name:atom, -OWLClass:term)
%
% @param Name is the 2nd argument of the object-condition
% @param OWLClass is a named class (possibly owl:Thing)
%
get_namedclass(somebody, 'OWLClass'('owl:Thing')) :- !.
get_namedclass(something, 'OWLClass'('owl:Thing')) :- !.
get_namedclass(thing, 'OWLClass'('owl:Thing')) :- !.
get_namedclass(NamedClass, 'OWLClass'(NamedClass)).
```

`make_restr/5` builds an object property restriction from a numerical quantifier type, numerical quantifier number, OWL property description, and OWL class description.

```
%% make_restr(+QType:atom, +QNum:atomic, +Property:term, +Class:term,
%%            -Restriction:term)
%
% @param QType is in {na, eq, geq, leq, greater, less}
% @param QNum is a positive integer or 'na' (not available)
% @param Property is a property description
% @param Class is a class description
% @param Restriction is a class description built from
%                    QType, QNum, Property, and Class
%
```

```
make_restr(na, na, Property, Class, 'ObjectSomeValuesFrom'(Property, Class)).
make_restr(eq, 1, Property, Class, 'ObjectSomeValuesFrom'(Property, Class)).
make_restr(geq, 1, Property, Class, 'ObjectSomeValuesFrom'(Property, Class)).
make_restr(eq, QNum, Property, Class,
              'ObjectMinCardinality'(QNum, Property, Class)) :- QNum > 1.
make_restr(geq, QNum, Property, Class,
              'ObjectMinCardinality'(QNum, Property, Class)) :- QNum > 1.
make_restr(leq, QNum, Property, Class,
               'ObjectMaxCardinality'(QNum, Property, Class)).
make_restr(less, QNum, Property, Class,
          'ObjectMaxCardinality'(Num, Property, Class)) :- Num is QNum - 1.
make_restr(greater, QNum, Property, Class,
          'ObjectMinCardinality'(Num, Property, Class)) :- Num is QNum + 1.
```

`is_chain/4` builds an ordered list of OWL property descriptions from a list of DRS predicate-conditions. The arguments of the predicates must only map to object-conditions that have been derived from ACE indefinite pronouns ('somebody', 'something', ...). The elements of the list are linked to each other via argument-sharing so that a "chain" is formed.

```
%% is_chain(+Ref1:nvar, +RefN:nvar, +CondList:list, -SubPropertyChain:list)
%
% @param Ref1 is a discourse referent of the first object in the chain
% @param RefN is a discourse referent of the last object in the chain
% @param CondList is a list of DRS conditions
% @param SubPropertyChain is a list of property descriptions in chain-order
%
is_chain(Ref, Ref, [], []).

is_chain(Ref1, RefN, CondList, ['ObjectProperty'(Property) | Chain]) :-
        select(predicate(_, Property, Ref1, Tmp)-_, CondList, CondList1),
        Property \= be,
        has_dom_for_member(Tmp, CondList1, CondList2),
        is_chain(Tmp, RefN, CondList2, Chain).

is_chain(Ref1, RefN, CondList,
              ['InverseObjectProperty'('ObjectProperty'(Property)) | Chain]) :-
        select(predicate(_, Property, Tmp, Ref1)-_, CondList, CondList1),
        Property \= be,
        has_dom_for_member(Tmp, CondList1, CondList2),
        is_chain(Tmp, RefN, CondList2, Chain).
```

`has_dom_for_member/3` consumes an object-condition from a list of conditions, such that the object-condition corresponds the an ACE indefinite pronoun.

```
%% has_dom_for_member(+Ref:nvar, +CondListIn:list, -CondListOut:list)
%
% @param Ref is a discourse referent
% @param CondListIn is a list of DRS conditions
% @param CondListOut is the remaining list of DRS conditions (after select/3)
%
has_dom_for_member(Ref, CondListIn, CondListOut) :-
```

```
    select(object(Ref, Name, Count, _, QType, QNum)-_, CondListIn, CondListOut),
    (
            Name = something, Count = dom
    ;
            Name = somebody, Count = countable
    ;
            Name = thing, Count = countable, QType = eq, QNum = 1
    ).
```

Implications that map to *SubClassOf*

When building the *SubClassOf*-axiom, the goal is to roll up the condition lists in the *if*-box and the *then*-box given the constraints listed in section 5.5.4 and given the possibility to invert the subject and object arguments the predicate-conditions. If this goal fails then the implication cannot be mapped to *SubClassOf*.

```
%% condition_axiom(+Condition:term, +RefList:list, -Axiom:term)
%
% @param Condition is a DRS condition
% @param RefList is a list of top-level discourse referents
% @param Axiom is an OWL axiom
%
condition_axiom(
        If => Then,
        RefList,
        'SubClassOf'(IfClass, ThenClass)
) :-
        condlist_if(X, If, RefList, IfClass),
        condlist_and(X, Then, RefList, ThenClass).
```

where `condlist_if/4` establishes the distinguished referent of the implication (i.e. the root of the tree) and then calls `condlist_and/4`. `condlist_and/4` calls `condlist_classlist/4` and if the latter returns a 1-element class-list then just this single element is returned (because the OWL syntax does not allow 1-element intersections).

`condlist_classlist/4` builds a list of OWL class descriptions from a list of DRS-conditions, i.e. the returned list must be interpreted as an intersection. The elements in the list must share the distinguished referent.

```
%% condlist_classlist(+D:nvar, +CondList:list, +RefList:list, -ClassList:list)
%
% @param D is a discourse referent
% @param CondList is a list of DRS conditions
% @param RefList is a list of top-level discourse referents
% @param ClassList is a list of class descriptions
%
condlist_classlist(_D, [], _RefList, []).

condlist_classlist(D, CondList, RefList, [And1Class | And2Class]) :-
        select(Condition, CondList, CondList1),
        condlist_class(D, Condition, CondList1, RefList, And1Class, CondList2),
        condlist_classlist(D, CondList2, RefList, And2Class).
```

`condlist_class/5` is the main (and most complex) rule in the mapping. Each alternative rule consumes a predicate-condition with its connected object-conditions, a negation-box or disjunction-boxes (with their contents), and converts the list of consumed conditions into an OWL class description, returning in addition the remaining (unconsumed) conditions.

The predicate-condition either corresponds to the copula verb 'be' or to a regular transitive verb. Both subject-object orders are allowed — for the copula verb, the order plays no role for the outcome because 'be' is interpreted as symmetric, for the transitive verbs, if the object matches the distinguished referent `D` then *InverseObjectProperty*-description is generated, otherwise no inverting is made. In case the subject and object arguments refer to the same discourse referent, then an *ObjectExistsSelf*-description is generated.

```
%% condlist_class(+D:nvar, Condition:term, +CondListIn:list, +RefList:list,
%%                +Class:term, -CondListOut:list)
%
% @param D is a discourse referent
% @param Condition is a DRS condition
% @param CondListIn is a list of DRS conditions
% @param RefList is a list of top-level discourse referents
% @param Class is a class description
% @param CondListOut is a list of remaining DRS conditions

% Copula ('be') predicate
% Every man is a human.
condlist_class(D, predicate(_, be, Subj, Obj)-_, CondList, RefList,
                                                    EmbeddedClass, CondList2) :-
    (
            Subj = D, Obj = NewD
    ;
            Obj = D, Subj = NewD
    ),
    select_object(NewD, CondList, RefList, NamedClass, QType, QNum, CondList1),
    (
            QType = eq, QNum = 1
    ;
            QType = na, QNum = na
    ),
    follow_object(NewD, NamedClass, CondList1, RefList, EmbeddedClass, CondList2).

% Regular predicate with reflexive object
% Every man likes himself.
condlist_class(D, predicate(_, Property, D, D)-_, CondList, _RefList,
                     'ObjectExistsSelf'('ObjectProperty'(Property)), CondList) :-
    Property \= be.

% Regular predicate with a dist. variable
% Every man likes a woman.
condlist_class(D, predicate(_, PropertyName, D, Obj)-_, CondList, RefList,
                                                          Class, CondList2) :-
    PropertyName \= be,
```

```
    select_object(Obj, CondList, RefList, NamedClass, QType, QNum, CondList1),
    follow_object(Obj, NamedClass, CondList1, RefList, EmbeddedClass, CondList2),
    make_restr(QType, QNum, 'ObjectProperty'(PropertyName), EmbeddedClass, Class).

% Regular predicate with a dist. variable, inverted case
% Every man is liked by a woman.
condlist_class(D, predicate(_, PropertyName, Subj, D)-_, CondList, RefList,
                                                          Class, CondList2) :-
    PropertyName \= be,
    select_object(Subj, CondList, RefList, NamedClass, QType, QNum, CondList1),
    follow_object(Subj, NamedClass, CondList1, RefList, EmbeddedClass, CondList2),
    make_restr(QType, QNum, 'InverseObjectProperty'('ObjectProperty'(PropertyName)),
                            EmbeddedClass, Class).
```

Negations and disjunctions that are embedded in the implication-condition are constrained by the requirements given above, i.e. the distinguished referent of the negation-box must match the distinguished referent of the outer box and same holds for the disjunction-boxes which in addition must share the distinguished referent among each other.

```
% Negation
% Every man is not a table.
condlist_class(D, -Not, CondList, RefList,
                            'ObjectComplementOf'(NotClass), CondList) :-
    condlist_and(D, Not, RefList, NotClass).

% Disjunction
% Every man is policeman or is not a table.
condlist_class(D, Or1 v Or2, CondList, RefList,
                            'ObjectUnionOf'([Or1Class, Or2Class]), CondList) :-
    condlist_and(D, Or1, RefList, Or1Class),
    condlist_and(D, Or2, RefList, Or2Class).
```

The rule `select_object/7` takes a discourse referent as an argument and either returns the *ObjectOneOf*-description (if the referent is a top-level referent), or returns the lemma of the object-condition (if the referent corresponds to an object-condition).

```
%% select_object(+D:nvar, +CondListIn:list, +RefList:list, -NamedClass:term,
%%               -QType:atom, -QNum:atomic, -CondListOut:list)
%
% @param D is a distinguished discourse referent
% @param CondListIn is a list of DRS conditions
% @param RefList is a list of top-level discourse referents
% @param NamedClass is either OWLClass or ObjectOneOf([_])
% @param QType is in {na, eq, geq, leq, greater, less}
% @param QNum is a positive integer or 'na' (not available)
% @param CondListOut is a list of remaining DRS conditions
%
select_object(D, CondList, RefList, 'ObjectOneOf'([Individual]), eq, 1, CondList) :-
    is_toplevel(D, RefList, 'ObjectOneOf'([Individual])).

select_object(D, CondListIn, _RefList, NamedClass, QType, QNum, CondListOut) :-
```

```
select(object(D, Name, _, _, QType, QNum)-_, CondListIn, CondListOut),
get_namedclass(Name, NamedClass).
```

The rule `follow_object/6` builds a complex class description based on the (distinguished) discourse referent given as an argument. The complexity of this rule derives from the fact that in the DRS, the intersection is represented as co-occurrence of conditions in the same DRS-box, i.e. there is no dedicated box for intersection as there is for negation and disjunction. In OWL, on the other hand, intersection has a dedicated constructor, and therefore descriptions where intersection is embedded inside property restrictions have more structure than the corresponding representation in the DRS. The mapping algorithm tries to build class descriptions undeterministically, i.e. if building fails then the algorithm backtracks and tries another way.

The algorithm tries to build a complete class-list and consume all the conditions.

1. If this succeeds then the returned class-list is either empty or not.
   (a) In the case of an empty class-list, `Class` is simply a named class.
   (b) In the case of a non-empty class-list, `Class` is an intersection with the named class as the first element.
2. If building the complete class-list fails (i.e. some conditions were not consumed) then an attempt is made to build just one class description.
   (a) If this succeeds then it might be possible to build more class descriptions, i.e. we call `follow_object/6` recursively.
      i. If this fails then we return an intersection of the named class and the newly built class description, along with all the unconsumed conditions.
   (b) If building just one class description fails, then we return the named class along with the unconsumed conditions.

`follow_object/6` is formally expressed in the following way.

```
%% follow_object(+D:nvar, +NamedClass:atom, +CondListIn:list, +RefList:list,
%%               -Class:term, -CondListOut:list)
%
% @param D is a distinguished discourse referent
% @param NamedClass is either OWLClass or ObjectOneOf([_])
% @param CondListIn is a list of DRS conditions
% @param RefList is a list of top-level discourse referents
% @param Class is a class description
% @param CondListOut is a list of remaining DRS conditions
%
follow_object(D, NamedClass, CondList, RefList, Class, CondListOut) :-
  (
      condlist_classlist(D, CondList, RefList, ClassList)
  ->
      (
          ClassList = []
```

```
      ->
          Class = NamedClass
      ;
          Class = 'ObjectIntersectionOf'([NamedClass | ClassList])
      ),
      CondListOut = []
  ;
      (
          (
              select(Condition, CondList, CondList1),
              condlist_class(D, Condition, CondList1, RefList, Class1, CondList2)
          )
      ->
          (
              follow_object(D, 'OWLClass'('owl:Thing'), CondList2, RefList,
                                    Class2, CondList3)
          ->
              Class = 'ObjectIntersectionOf'([NamedClass, Class1, Class2]),
              CondListOut = CondList3
          ;
              Class = 'ObjectIntersectionOf'([NamedClass, Class1]),
              CondListOut = CondList2
          )
      ;
          Class = NamedClass,
          CondListOut = CondList
      )
  ).
```

Implications that map to *SubObjectPropertyOf* and *DisjointObjectProperties* axioms

The FOL-representation of OWL property axioms has a very fixed pattern. We therefore need to apply a simple DRS-template to capture either the *SubObjectPropertyOf* or the *DisjointObjectProperties* axiom.

SubObjectPropertyOf must have a (non-empty) property chain in the *if*-part of the implication and a single predicate-condition in the *then*-part. Furthermore, the start-point and the end-point of the chain are forced to be the same as the subject and object arguments of the predicate-condition in the *then*-part. OWL 1.1 *SubObjectPropertyOf*-axiom can be seen as a generalization of OWL DL's subproperty and transitivity axioms. In order to be more compatible with OWL DL tools, we map the general *SubObjectPropertyOf*-axiom to the specific syntax for transitivity axioms and simple subproperty axioms, if applicable. This is performed by `convert_to_owldl/2` which is a straightforward mapping and not shown here.

```
condition_axiom(
        If => [predicate(_, Property, Ref1, RefN)-_],
        _,
        PropertyAxiom
) :-
        Property \= be,
        has_dom_for_member(Ref1, If, CondList1),
```

```
        is_chain(Ref1, RefN, CondList1, SubPropertyChain),
        SubPropertyChain = [_ | _],
        convert_to_owldl(
                'SubObjectPropertyOf'(
                        'SubObjectPropertyChain'(SubPropertyChain),
                        'ObjectProperty'(Property)
                ),
                PropertyAxiom
        ).
```

DisjointObjectProperties are generated by requiring a 1-element chain (i.e. a predicate-condition with two indefinite object-conditions linked to the predicate as arguments) in the *if*-part and a negated predicate-condition in the *then*-part. Similarly to *SubObjectPropertyOf*, the start-point and the end-point of the chain must be the same as the subject and object arguments of the predicate-condition in the *then*-part. Note that in OWL, *DisjointObjectProperties* has an argument that is a set with any number of elements. From the DRS, we can only generate 2-element *DisjointObjectProperties*-sets, i.e. the semantics of *n*-element *DisjointObjectProperties*-sets must be captured by a set of 2-element *DisjointObjectProperties*-sets.

```
condition_axiom(
        If => [-[predicate(_, Property2, Ref1, RefN)-_]],
        _,
        'DisjointObjectProperties'([Property1, 'ObjectProperty'(Property2)])
) :-
        Property2 \= be,
        has_dom_for_member(Ref1, If, CondList1),
        is_chain(Ref1, RefN, CondList1, [Property1]).
```

Top-level DRS

The top-level DRS is relatively easy to handle. Each top-level condition is mapped to a corresponding OWL axiom using the `condition_axiom/3` rule.

Named-object-condition (i.e. the condition derived from proper names) is mapped as follows.

```
condition_axiom(
        object(_, ProperName, named, _, eq, 1)-_,
        _,
        'ClassAssertion'('Individual'(ProperName), 'OWLClass'('owl:Thing'))
).
```

Other object-conditions are mapped to anonymous individuals identified by `Ref` and typed by the named class `NamedClass`. The arguments of the object-condition are checked so that this rule would not fire on conditions derived from noun which are number-restricted.

```
condition_axiom(
        object(Ref, Name, Count, _, QType, QNum)-_,
```

```
        _,
        'ClassAssertion'('AnonymousIndividual'(Ref), NamedClass)
) :-
        member(Count, [countable, dom]),
        member(QType, [eq, geq, na]),
        member(QNum, [1, na]),
        get_namedclass(Name, NamedClass).
```

The copula-predicate-condition maps to the *SameIndividual*-axiom.

```
condition_axiom(
        predicate(_, be, Ref1, Ref2)-_,
        RefList,
        'SameIndividual'([Name1, Name2])
) :-
        is_toplevel(Ref1, RefList, 'ObjectOneOf'([Name1])),
        is_toplevel(Ref2, RefList, 'ObjectOneOf'([Name2])).
```

All the remaining predicate-conditions map to the *ObjectPropertyAssertion*-axiom.

```
condition_axiom(
        predicate(_, Predicate, Ref1, Ref2)-_,
        RefList,
        'ObjectPropertyAssertion'('ObjectProperty'(Predicate), Name1, Name2)
) :-
        Predicate \= be,
        is_toplevel(Ref1, RefList, 'ObjectOneOf'([Name1])),
        is_toplevel(Ref2, RefList, 'ObjectOneOf'([Name2])).
```

Disjunction and negation, however, need a more complex treatment. Namely, we call `condlist_and/4` to map either the negation box of disjunction boxes to an OWL class description. Note that the distinguished referent must be shared with the top-level box and must be shared between the disjoined boxes.

Mapping of top-level negation-conditions, derived from e.g. sentences like "John does not like a man who owns at most 2 cars." which maps to the axiom $\{John\} \sqsubseteq \neg(\exists\, like(man \sqcap (\leq 2\, own\, car)))$.

```
condition_axiom(
        -Not,
        RefList,
        'ClassAssertion'(Individual, 'ObjectComplementOf'(NotClass))
) :-
        condlist_and(D, Not, RefList, NotClass),
        is_toplevel(D, RefList, 'ObjectOneOf'([Individual])).
```

Mapping of top-level disjunction-conditions, derived from sentences like "John likes Mary or does not like Bill." which maps to the axiom $\{John\} \sqsubseteq (\exists\, like\, \{Mary\}) \sqcup \neg(\exists\, like\, \{Bill\})$.

```
condition_axiom(
        Or1 v Or2,
        RefList,
        'ClassAssertion'(Individual, 'ObjectUnionOf'([Or1Class, Or2Class]))
) :-
        condlist_and(D, Or1, RefList, Or1Class),
        condlist_and(D, Or2, RefList, Or2Class),
        is_toplevel(D, RefList, 'ObjectOneOf'([Individual])).
```

5.5.7 Error messages

The ACE parser developed in the Attempto project offers various error messages in case the input text violates the syntax of ACE. For example:

- Add a determiner before: 'cat' (in the case of "Mary owns cat.");
- The construct 'to' followed by a verb is not allowed. Please replace it by 'that' and a subordinated sentence (in the case of "John likes to eat.");
- Verb cannot be used as singular intransitive: 'meets' (in the case of "John meets.").

Because the conversion from ACE to OWL accepts only a subset of ACE as its input, additional error messages have been designed to account for the input texts that are not in this subset. Although the input to the translator is really a DRS, the error messages refer to ACE constructs instead of the DRS constructs as most users of the translator work on the ACE level and the preprocessing step of translating ACE texts into the DRS language takes place in the background, hidden from the user. In most cases, the DRS conditions contain the content words of the original text and include sentence identifiers. This information is reported back to the user for better localization of the error.

There are two kinds of error messages. The first kind of messages (table 5.1) are triggered by DRS conditions which the ACE→OWL mapping does not support. For example, conditions corresponding to noun and verb modifiers which are not included in ACE_1. The full list of such constructs was briefly described in section 2.3.6.

The second kind of messages (table 5.2) are about illegal argument sharing between DRS conditions. They generally point to cases where anaphoric references have been used in such a way that rolling up fails, i.e. the referencing (or lack of it) goes beyond a tree structure and thus cannot be expressed in OWL (see section 5.5.4). The messages report the problematic sentences and pinpoint the exact verbs or nouns that are the source of the problem.

In the first case, resolving the error is simple — the user must rephrase the sentence without using the violating word or construct. In the second case, the user must give up the violating references or provide the missing ones. In order to avoid the second kind of errors, the users can be advised to express all universally quantified sentences as *every*-sentences without any explicit anaphoric references apart from proper names and reflexive pronouns. The fragment of ACE (called ACE_2) that obeys those restrictions is used also when verbalizing OWL ontologies and is discussed in detail in section 5.6.2.

Unsupported DRS condition	Error message
□ DRS	Necessity not supported
◇ DRS	Possibility not supported
∼ DRS	Negation-as-failure not supported
Label:DRS	Sentence subordination not supported
modifier_adv(_, *Adverb*, _)-Id	Adverbs not supported: *Adverb*
modifier_pp(_, *Preposition*, _)-Id	Prepositional phrases not supported: *Preposition*
object(_, na, _, _, _, _)-Id	Noun phrase conjunctions not supported: and
property(_, *Adjective*, _)-Id	Adjectives not supported: *Adjective*
property(_, *Adjective*, _, _)-Id	Adjectives not supported: *Adjective*
property(_, *Adjective*, _, _, _, _)-Id	Adjectives not supported: *Adjective*
query(_, *QueryWord*)-Id	Queries not supported: *QueryWord*
predicate(_, *Verb*, _)-Id	Intransitive verbs not supported: *Verb*
predicate(_, *Verb*, _, _, _)-Id	Ditransitive verbs not supported: *Verb*

Table 5.1: Error messages about illegal DRS conditions. Note that the error messages are reported on the ACE level, showing the identifier of the violating sentence and sometimes the word with its part of speech (e.g. adjective, intransitive verb).

Error message example	Example of erroneous ACE sentence
'woman': A reference to this noun either does not exist or is illegal.	If a man owns a car then there is a woman.
'bite': Subject or object of this verb makes an illegal reference.	Every man who owns a cat hates a dog that bites the cat.

Table 5.2: Error messages about illegal argument sharing. In the first sentence, the *if* and *then* parts share no objects, i.e. the *if*-part introduces an object 'man' but the *then*-part does not refer to it, instead it introduces a new and "unconnected" object 'woman'. In the second sentence, there is too much interlinking between the *if* and *then* parts — the *every*-quantified noun phrase "man who owns a cat" (which corresponds to the *if*-part) introduces two objects, 'man' and 'cat', and the verb phrase "hates a dog that bites the cat" (which corresponds to the *then*-part) makes an implicit reference to the 'man' and an explicit reference to the 'cat'.

5.5.8 Incompleteness

In this section, we describe two kinds of incompleteness of the ACE→OWL algorithm. First, there exist certain ACE sentences that have a natural counterpart in OWL, but which are not translated by our algorithm. Examples of such sentences (also discussed in section 2.5.4) are:

(5.31) If a man owns a car X and owns a car Y then X is Y. (which could be translated in the same way as: "Every man owns at most 1 car.")

(5.32) If somebody X writes something Y then X is an author and Y is a book. (which could be translated in the same way as: "Everybody that writes something is an author. Everything that somebody writes is a book.")

Applying more rewriting to the input DRS would solve some of those incompleteness issues, but on the other hand, such rewriting could also be done explicitly by the user. Thus we argue that the completeness property is not so important in this case. However, if it turns out that sentences such as given in the previous example occur often in practice, then the translation algorithm must be extended.

The second kind of incompleteness has to do with generating all possible OWL expressions. It is clear that we cannot (and have not set out to) generate all OWL's syntactic forms, e.g. the translation algorithm never generates the *ObjectAllValuesFrom*-description. The question is rather whether we can capture the semantics of all OWL axioms.

Consider that all OWL class descriptions have a tree structure. This structure can be flattened and represented as a list of sentences (either in the *if*-part or the *then*-part of the containing sentence). References to class names would be made in this case by definite noun phrases or explicit variables. Negation, however, complicates the picture since we would have to use the sentence negation 'it is false that' which does not mark the beginning and end of its scope explicitly. Thus it is impossible to embed two negations into one (on the same level). For example, the sentence

(5.33) it is false that it is false that S_1 and that it is false that S_2

can be interpreted in two ways, namely

(5.34) it is false that ((it is false that S_1) and that (it is false that S_2))

(5.35) it is false that (it is false that (S_1 and that it is false that S_2))

In ACE, the sentence 5.33 expresses only the reading 5.35 and it is not possible to change the sentence in order to express the reading 5.34. Thus, very complex class descriptions cannot be completely flattened as we would still need (in addition to explicit variables) some form of parentheses. Again, as such constructions do not occur in practice and are better avoided anyway, we consider this incompleteness not an issue.

5.6 Verbalizing OWL in ACE

5.6.1 Introduction

In section 5.5, we discussed the mapping of an ACE fragment ACE_1 to OWL. In this fragment, there exist syntactically different possibilities to express a certain OWL axiom. Also, as explained in section 5.5.7, ACE_1 allows statements that go beyond OWL in terms of expressivity. In this section, we confine ourselves to a subset of ACE_1, which we call ACE_2 and from which all syntactic variety has been removed, i.e. every DRS that is supported by this fragment can be expressed in only one way. We find this fragment particularly suitable for verbalizing (and also writing) OWL ontologies because the axioms are expressed in compact *every*-sentences which furthermore offer the user no opportunity to express variable patterns which OWL would not be able to express. I.e. this fragment makes explicit which sentences are compatible with OWL's expressivity and which are not.

In the following, section 5.6.2 describes the grammar of ACE_2 by listing the restrictions on the grammar of ACE_1. Section 5.6.3 describes ACE_2 formally by a Definite Clause Grammar (DCG) that also provides a mapping of ACE_2 to (a fragment of) OWL. As this DCG supports only some of OWL's syntactic patterns, sections 5.6.4 and 5.6.5 show how all OWL expressions (with a few exceptions) can be reduced to the DCG-supported fragment. Finally, section 5.6.6 summarizes the complete verbalization algorithm composed of the "modules" described before.

5.6.2 ACE_2 construction rules

ACE_2 is defined by a few clear-cut restrictions on the syntax of ACE_1. The main motivation for those restrictions is a need to avoid variable sharing patterns which in OWL are not expressible. A natural (and easy to learn) solution is to remove all forms of anaphoric references from ACE_2, thus arriving at a "variable-free syntax". Such syntax excludes sentences like

(5.36) Every man who owns a cat hates a dog that bites **the cat**.

In addition, we must add a restriction that excludes sentences that avoid variable sharing, such as

(5.37) If a man owns a car then there is a woman.

Here the solution is to remove *if-then* sentences from ACE_2.

As a result, we are left with *every*-sentences which can only achive variable sharing patterns that are precisely the ones supported by OWL.

Still, for two reasons, these restrictions must be relaxed — first, OWL supports the *ObjectExistsSelf*-description which maps naturally to reflexive pronouns like 'itself' (which are anaphoric); secondly, property axioms (and especially property chains) include variable sharing which cannot be expressed by relative clauses only. Therefore, we allow reflexive pronouns and certain *if-then* sentence templates.

The rest of the restrictions on noun phrases, verb phrases and sentences mainly remove syntactic sugar, and by that, reflect what, in our opinion, is the best choice of function words for verbalizing OWL ontologies, e.g. we do not allow negated numerical quantifiers like 'not at least 2' because they are hard to read.

Noun phrases

In ACE_2, only the following determiners are allowed: 'every', 'no', 'a', 'nothing but', 'at least n', 'at most n', 'exactly n'. Indefinite pronouns are restricted to 'everything', 'nothing' and 'something', and reflexive pronouns to 'itself'. Variables are allowed, but they can be used only in fixed sentence patterns.

Relative clauses cannot be object relative clauses. Furthermore, proper names and reflexive pronouns cannot be modified by relative clauses.

(5.38) * John that owns a car

(5.39) * itself that owns a car

Verb phrases

In ACE_2, verb phrases cannot be coordinated, i.e. all coordination must be written as relative clause coordination. In ACE_2, verbs phrase objects cannot be universally quantified, i.e. 'every man', 'no man', 'everything', 'nothing' are allowed as subjects, but not as objects.

(5.40) * Every dog hates every cat.

(5.41) * John knows every customer.

Also, a proper name cannot be used as a subject in passive constructions.

(5.42) * John is liked by Mary.

Sentences

In general, every ACE_2 sentence must start with a universally quantified noun phrase or a proper name, contain a verb phrase, and end with a full-stop.

(5.43) No man inserts a card.

(5.44) Every man is a human.

(5.45) John inserts a card.

(5.46) * A man is a human.

(5.47) * If a man inserts ...

Still, two *if-then* sentence templates are supported which violate this general rule. Those sentence templates also allow for a restricted use of anaphoric references to preceding noun phrases. The templates are as follows

(5.48) If something X R_1 something that R_2 something that ... something that R_n something Y then X S Y.

(5.49) If something X R something Y then it is false that X S Y.

where $R_1, \ldots, R_n$, S are ACE transitive verbs in either active or passive, and X and Y are ACE variables. An example of a sentence that follows pattern 5.48 is

(5.50) If something X lives-in something that is contained by something Y then X lives-in Y.

5.6.3 Formal grammar of ACE_2

In this section, we show a mapping which transforms an OWL axiom (in a fragment of OWL) into an ACE sentence (in the ACE_2 fragment). This mapping is given by a bidirectional Prolog Definite Clause Grammar.

Content words

Being consistent with the ACE→OWL mapping described earlier, named OWL classes map to ACE nouns, named OWL properties and their inverses map to ACE transitive verbs, and *ObjectOneOf*-descriptions that contain exactly one individual name map to ACE proper names.

For nouns, we have to describe two forms: singular (*sg*) and plural (*pl*). The noun 'thing' is mapped to *owl:Thing*, other nouns map to class names, so that singular nouns are mapped to themselves and plural nouns are mapped to their lemmas (i.e. singular forms). Therefore, the singular form is used as the OWL class name. Formally, the mapping is given by the `n`-rule.

```
n(num=sg, 'OWLClass'('owl:Thing')) -->
        [thing].
n(num=pl, 'OWLClass'('owl:Thing')) -->
        [things].
n(num=sg, 'OWLClass'(Token)) -->
        [Token].
n(num=pl, 'OWLClass'(TokenSg)) -->
        [TokenPl],
        { noun_pl(TokenSg, TokenPl) }.
```

where `noun_pl/2` is a bidirectional mapping of singular nouns to plural nouns.

For verbs we have to describe 8 forms, determined by the dimensions *number*, *being negated*, and *being in past participle* as

$$\{\text{num=sg}, \text{num=pl}\} \times \{\text{neg=yes}, \text{neg=no}\} \times \{\text{vbn=yes}, \text{vbn=no}\}$$

Examples of verb forms are thus: ‘modifies’, ‘modify’, ‘does not modify’, ‘do not modify’, ‘is modified by’, ‘are modified by’, ‘is not modified by’, ‘are not modified by’. The lexicon, however, must provide only three forms: finite form, infinitive, and past participle (e.g. ‘modifies’, ‘modify’, ‘modified’), and map the finite and participle forms to the infinitive to be used as OWL’s property name. (Note that alternatively we could use the finite form as the property name.) Formally, the mapping is given by the `tv`-rule.

```
tv(num=sg, neg=no, vbn=no, 'ObjectProperty'(TokenInf)) -->
        [TokenSg],
        { verb_sg(TokenInf, TokenSg) }.
tv(num=pl, neg=no, vbn=no, 'ObjectProperty'(Token)) -->
        [Token].
tv(num=sg, neg=yes, vbn=no, 'ObjectProperty'(Token)) -->
        [Token].
tv(num=pl, neg=yes, vbn=no, 'ObjectProperty'(Token)) -->
        [Token].
tv(_, _, vbn=yes, 'InverseObjectProperty'('ObjectProperty'(TokenInf))) -->
        [TokenPp, by],
        { verb_pp(TokenInf, TokenPp) }.
```

where `verb_sg/2` is a bidirectional mapping of infinite verbs to 3rd person singular verbs, and `verb_pp/2` is a bidirectional mapping of infinite verbs to past participles.

Proper names do not have to be included in the lexicon as they do not undergo any morphological changes, e.g. we do not allow plural proper names. Still, we must avoid the overlap with function words ‘itself’ and ‘themselves’. Formally, the mapping is given by the `propn`-rule.

```
propn('ObjectOneOf'(['Individual'(ProperName)])) -->
        [ProperName],
        { ProperName \= itself, ProperName \= themselves }.
```

Function words

On the OWL level, we distinguish between two *SubClassOf*-axioms, the first expresses disjointness of classes, the second the general subclass-relationship. On the ACE level, there are two kinds of determiners, subject determiners and object determiners. Subject determiners (`det_subj`) ’Every’ and ‘No’ are allowed only to precede subjects (and consequently the sentence-initial words).

```
det_subj(C1, C2, 'SubClassOf'(C1, 'ObjectComplementOf'(C2))) -->
        ['No'].

det_subj(C1, C2, 'SubClassOf'(C1, C2)) -->
        ['Every'].
```

Object determiners (`det_obj`) are allowed with nouns that act as objects. These determiners decide the type of the property restriction.

```
det_obj(num=sg, R, C, 'ObjectSomeValuesFrom'(R, C)) -->
        [a].
det_obj(num=pl, R, C, 'ObjectAllValuesFrom'(R, C)) -->
        [nothing, but].
det_obj(num=sg, R, C, 'ObjectMinCardinality'(1, R, C)) -->
        [at, least, 1].
det_obj(num=sg, R, C, 'ObjectMaxCardinality'(1, R, C)) -->
        [at, most, 1].
det_obj(num=sg, R, C, 'ObjectExactCardinality'(1, R, C)) -->
        [exactly, 1].
det_obj(num=pl, R, C, 'ObjectMinCardinality'(Integer, R, C)) -->
        [at, least, Integer],
        { between(2, infinite, Integer) }.
det_obj(num=pl, R, C, 'ObjectMaxCardinality'(Integer, R, C)) -->
        [at, most, Integer],
        { between(2, infinite, Integer) }.
det_obj(num=pl, R, C, 'ObjectExactCardinality'(Integer, R, C)) -->
        [exactly, Integer],
        { between(2, infinite, Integer) }.
```

Some rules use the Prolog built-in `between(+Low, +High, ?Value)` to accept or generate integers between 2 and ∞.

The `auxc`-rule handles singular and plural copula verbs, with or without negation, where ACE's negation maps to OWL's *ObjectComplementOf*.

```
auxc(num=sg, C, C) -->
        [is].
auxc(num=pl, C, C) -->
        [are].
auxc(num=sg, C, 'ObjectComplementOf'(C)) -->
        [is, not].
auxc(num=pl, C, 'ObjectComplementOf'(C)) -->
        [are, not].
```

The `auxv`-rule handles auxiliary verbs ('is', 'are', 'do', 'does') that are used with transitive verbs. Again, ACE's negation maps to OWL's *ObjectComplementOf*.

```
auxv(num=sg, neg=yes, vbn=no, C, 'ObjectComplementOf'(C)) -->
        [does, not].
auxv(num=pl, neg=yes, vbn=no, C, 'ObjectComplementOf'(C)) -->
        [do, not].
auxv(num=sg, neg=yes, vbn=yes, C, 'ObjectComplementOf'(C)) -->
        [is, not].
auxv(num=pl, neg=yes, vbn=yes, C, 'ObjectComplementOf'(C)) -->
        [are, not].
auxv(num=sg, neg=no, vbn=yes, C, C) -->
        [is].
auxv(num=pl, neg=no, vbn=yes, C, C) -->
        [are].
auxv(_, neg=no, vbn=no, C, C) -->
        [].
```

OWL class	Corresponding ACE noun phrase
ObjectComplementOf(cat)	something that is not a cat
ObjectIntersectionOf(cat car ...)	something that is a cat and that is a car and that ...
ObjectUnionOf(cat car ...)	something that is a cat or that is a car or that ...
ObjectSomeValuesFrom(like cat)	something that likes a cat
ObjectAllValuesFrom(like cat)	something that likes nothing but cats
ObjectExistsSelf(like)	something that likes itself
ObjectMinCardinality(2 like cat)	something that likes at least 2 cats
ObjectMaxCardinality(2 like cat)	something that likes at most 2 cats
ObjectExactCardinality(2 like cat)	something that likes exactly 2 cats

Table 5.3: Example of verbalizing OWL class descriptions as ACE noun phrases. Note that in actual verbalizations, the word 'something' is often replaced by a noun representing a conjoined named class, i.e. *ObjectIntersectionOf(cat ObjectExistsSelf(like))* is verbalized as "a cat that likes itself".

ObjectIntersectionOf and *ObjectUnionOf* are verbalized by ACE coordination words ', and' ("comma and"), 'and', and 'or'.

```
comma_and(C1, C2, 'ObjectIntersectionOf'([C1, C2])) -->
        [',', and].
and(C1, C2, 'ObjectIntersectionOf'([C1, C2])) -->
        [and].
or(C1, C2, 'ObjectUnionOf'([C1, C2])) -->
        [or].
```

Phrases

In OWL, it is possible to build complex class descriptions from simpler ones by intersection, union, complementation and property restriction. Similarly, ACE_2 allows building complex noun phrases via relative clauses that can be conjoined (by 'and that'), disjoined (by 'or that'), negated (by 'that is/are not, that does/do not') and embedded (by 'that'). See table 5.3 for examples.

Embedding allows us to use a relative clause to modify an object of another relative clause. For instance, the OWL class description that contains a complex class description within another complex class description

(5.51) $$cat \sqcap \neg(\exists like\,(dog \sqcap ((\exists attack\ mailman) \sqcup \{Fido\})))$$

can be verbalized in ACE_2 as

(5.52) something that is a cat and that does not like a dog that attacks a mailman or that is Fido

Class descriptions in OWL can be syntactically arbitrarily complex as one can use parentheses to denote the scope of the description. ACE, however, has no support for parentheses. Scope ambiguities are resolved according to the interpretation rules and the users have a choice between disentangling complex sentences, or using syntactic means to enforce the desired scoping. For instance, the binding order of *and* and *or* favors *and*, but can be reversed by using a comma in front of *and*. This approach is natural (as natural language does not use parentheses for grouping) but for the verbalization process it poses a problem as very complex class descriptions cannot be mapped to ACE noun phrases. For example, a relative clause can either modify the object (via 'that') or the subject (via 'and/or that') of a preceding relative clause, but not a more distant noun. We thus need to detect such class descriptions before verbalization, and not verbalize them (see section 5.6.5). In the following, we describe the supported phrases formally.

`ibar` corresponds to a verb phrase which is either a copula construction (`auxc` + `cop`) or is composed of a (possibly empty) auxiliary and a transitive verb phrase.

```
ibar(Num, C2) -->
        auxc(Num, C1, C2),
        cop(C1).

ibar(Num, C2) -->
        auxv(Num, Neg, Vbn, C1, C2),
        vp(Num, Neg, Vbn, C1).
```

The `cop`-rule handles noun phrases of the copula construction. Such noun phrases can either be proper names, indefinite noun phrases with a relative clause, and indefinite noun phrases without a relative clause.

```
cop('ObjectOneOf'([Individual])) -->
        propn('ObjectOneOf'([Individual])).

cop('ObjectIntersectionOf'([C1, C2])) -->
        [a],
        n(num=sg, C1),
        relcoord(num=sg, C2).

cop(C) -->
        [a],
        n(num=sg, C).
```

The `vp`-rule handles the regular verb phrases where a transitive verb is combined with a noun phrase object.

```
vp(Num, Neg, Vbn, Restriction) -->
        tv(Num, Neg, Vbn, R),
        np_obj(Num, R, Restriction).
```

Due to expressivity restrictions in OWL, we make a distinction between two types of noun phrases — those that can be used as subjects (`np_subj`) and those that can be used as objects (`np_obj`). The `np_subj`-rule covers proper names and universally quantified singular nouns, with or without a relative clause (coordination).

```
np_subj(C, 'SubClassOf'('ObjectOneOf'([Individual]), C)) -->
        propn('ObjectOneOf'([Individual])).

np_subj(Y, 'SubClassOf'(C, D)) -->
        det_subj(C, Y, 'SubClassOf'(C, D)),
        n(num=sg, C).

np_subj(Y, 'SubClassOf'(C, D)) -->
        det_subj('ObjectIntersectionOf'([X, Coord]), Y, 'SubClassOf'(C, D)),
        n(num=sg, X),
        relcoord(num=sg, Coord).
```

The `np_obj`-rule covers proper names, reflexive pronouns ('itself' and 'themselves'), and indefinite (existentially quantified and number-restricted) nouns with or without a relative clause.

```
np_obj(_, R, 'ObjectSomeValuesFrom'(R, 'ObjectOneOf'([Individual]))) -->
        propn('ObjectOneOf'([Individual])).

np_obj(num=sg, R, 'ObjectExistsSelf'(R)) -->
        [itself].

np_obj(num=pl, R, 'ObjectExistsSelf'(R)) -->
        [themselves].

np_obj(_, R, Restriction) -->
        det_obj(Num, R, C, Restriction),
        n(Num, C).

np_obj(_, R, Restriction) -->
        det_obj(Num, R, 'ObjectIntersectionOf'([C, Coord]), Restriction),
        n(Num, C),
        relcoord(Num, Coord).
```

A binary *ObjectIntersectionOf* maps to a relative clause which can contain further relative clauses. The multi-level rule assures the correct the binding order of 'and' and 'or' that can be overridden by ', and'.

```
relcoord(Num, 'ObjectIntersectionOf'(['ObjectUnionOf'(CL1),
                                      'ObjectUnionOf'(CL2)])) -->
        relcoord_1(Num, 'ObjectUnionOf'(CL1)),
        [',', and],
        relcoord_1(Num, 'ObjectUnionOf'(CL2)).

relcoord(Num, 'ObjectIntersectionOf'(['ObjectUnionOf'(CL1), C2])) -->
        relcoord_1(Num, 'ObjectUnionOf'(CL1)),
        [',', and],
        relcoord_2(Num, C2).

relcoord(Num, 'ObjectIntersectionOf'([C1, 'ObjectUnionOf'(CL2)])) -->
        relcoord_2(Num, C1),
        [',', and],
```

```
        relcoord_1(Num, 'ObjectUnionOf'(CL2)).

relcoord(Num, Coord) -->
        relcoord_1(Num, Coord).


relcoord_1(Num, 'ObjectUnionOf'([C1, C2])) -->
        relcoord_2(Num, C1),
        [or],
        relcoord_1(Num, C2).

relcoord_1(Num, Coord) -->
        relcoord_2(Num, Coord).


relcoord_2(Num, 'ObjectIntersectionOf'([C1, C2])) -->
        rel(Num, C1),
        [and],
        relcoord_2(Num, C2).

relcoord_2(Num, Coord) -->
        rel(Num, Coord).


rel(Num, X) -->
        [that],
        ibar(Num, X).
```

Sentence patterns

OWL axioms are mapped to ACE_2 sentences that come in four different forms: they are either

- *every*-sentences, i.e. they start with 'every' or 'everything'. This is the general case for *SubClassOf*-axioms;
- *no*-sentences, i.e. they start with 'no' or 'nothing'. This is the case when the 2nd argument of the *SubClassOf*-axiom is contained inside the *ObjectComplementOf*-description;
- sentences that start with a proper name. This is the case when the first argument of the *SubClassOf*-axiom is contained inside the *ObjectOneOf*-description; or
- *if-then* sentences that use variables to make anaphoric references. This is the case for axioms *SubObjectPropertyOf* and *DisjointObjectProperties*.

Some examples are shown in table 5.4.

Formally, in order to obtain the complete sentence, a singular noun phrase is concatenated with a singular verb phrase, and a period is added.

OWL axiom	Corresponding ACE sentence
SubClassOf(man human)	Every man is a human.
SubClassOf(man ObjectComplementOf(woman))	No man is a woman.
SubClassOf(ObjectOneOf(John) ObjectComplementOf(ObjectIntersectionOf(man ObjectMinCardinality(2 own car))))	John is not a man that owns at least 2 cars.
SubObjectPropertyOf(love like)	If something X loves something Y then X likes Y.
SubObjectPropertyOf(SubObjectPropertyChain(own contain) own)	If something X owns something that contains something Y then X owns Y.
DisjointObjectProperties(is-child-of is-spouse-of)	If something X is-child-of something Y then it is false that X is-spouse-of Y.

Table 5.4: Verbalizing OWL axioms as ACE sentences. Note that the property axioms use variables to make anaphoric references.

```
ip(SubClassOf) -->
        np_subj(Y, SubClassOf),
        ibar(num=sg, Y),
        ['.'].
```

The property axioms require explicit anaphoric references. We consider them more readable if expressed by *if-then*-sentences. The verbalization of property axioms is done by a set of simple templates. Here we show only the template for the simple *SubObjectPropertyOf*-axiom (which does not contain a property chain), and the *DisjointObjectProperties*-axiom. The rest of the templates are similar to those.

```
ip('SubObjectPropertyOf'('ObjectProperty'(R), 'ObjectProperty'(S))) -->
        ['If', something, 'X', RSg, something, 'Y', then, 'X', SSg, 'Y', '.'],
        { verb_sg(R, RSg), verb_sg(S, SSg) }.

ip('DisjointObjectProperties'(['ObjectProperty'(R), 'ObjectProperty'(S)])) -->
        ['If', something, 'X', RSg, something, 'Y', then,
                                    it, is, false, that, 'X', SSg, 'Y', '.'],
        { verb_sg(R, RSg), verb_sg(S, SSg) }.
```

Post-processing

In the verbalization direction, a simple post-processing step is added. In this step, we apply some simple transformations to "beautify" the sentences, e.g.:

- 'a thing' $\mapsto$ 'something',
- 'Every thing' $\mapsto$ 'Everything',
- 'a apple' $\mapsto$ 'an apple',

- period is connected to the final word of the sentence,
- comma is connected to its preceding word.

5.6.4 Rewriting OWL axioms

Many constructs in the OWL 1.1 Functional-Style Syntax can be seen as syntactic sugar — on the one hand their semantic equivalent can be expressed by other constructs, and on the other hand they make using the language (in this case the Functional-Style Syntax) easier. For example, the *FunctionalObjectProperty* is syntactically simple as it takes just one argument (a property description). This makes using the construct easier, e.g. a graphical user interface can express functional properties by a checkbox — for each property name there is a checkbox which is either "on" or "off". (Note however that the checkbox is ambiguous, as being "off" might either mean that the property is not functional, or that the functionality information is unknown, e.g. the latter is the case in Protégé.) Functionality of some property description R can also be expressed by a syntactically more complex construct $\top \sqsubseteq \leq 1\, R\, \top$, which says that "everything" is linked to at most 1 "thing" via the property R. While harder to read and write, this makes the semantics of functionality more explicit (e.g. that the cardinality is defined via $\leq$ and not via $=$), and avoids relying on the knowledge of the checkbox semantics. Also, it avoids using the word 'functional' which might be unfamiliar or ambiguous to the user. For example, WordNet 3.0 lists 6 synsets (sets of synonyms) that correspond to the adjective 'functional', none of which contains the sense of this word as it is used in mathematics. A similar situation exists with other OWL constructor names, e.g. the word 'range' (used in *ObjectPropertyRange*) has according to WordNet, 9 meanings as a noun and 8 meanings as a verb.

Because ACE does not offer function words like 'functional' and 'range' with any predefined logical meaning (they are just regular adjectives, nouns and verbs), we have decided to rewrite a set of OWL axioms and class descriptions via more basic axioms and class descriptions which can then be verbalized using the ACE function words. In the case of functionality, we would use words like 'everything', 'at most', '1', and 'thing'.

Table 5.5 shows the rewriting rules which while preserving the semantics of the input axioms map them to general *SubClassOf*, *SubObjectPropertyOf*, and *DisjointObjectProperties* axioms. I.e. the axioms are mapped to the fragment of OWL that was defined in the previous section.

Note that this rewriting generates a list of axioms from the following axioms: *EquivalentClasses*, *DisjointClasses*, *DisjointUnion*, *EquivalentObjectProperties*, *DisjointObjectProperties*, *InverseObjectProperties*, *SameIndividual*, *DifferentIndividuals*. The disadvantage of this is that the compactness of the input is lost and there is a blow-up in new axioms. We discuss this later in section 5.7.4.

5.6.5 Rewriting OWL *SubClassOf*-axioms

From the resulting axioms, *SubObjectPropertyOf* and *DisjointObjectProperties* can be directly handled by the simple templates presented in section 5.6.3 as part of the DCG.

OWL axiom	Equivalent OWL axiom(s)
EquivalentClasses($C_1 \ldots C_n$)	SubClassOf(C_1 C_2), SubClassOf(C_2 C_1), ...
DisjointClasses($C_1 \ldots C_n$)	SubClassOf(C_1 ObjectComplementOf(C_2)), ...
DisjointUnion(D $C_1 \ldots C_n$)	*Rewriting via* SubClassOf, ObjectComplementOf and ObjectUnionOf. (Discussed further in section 5.7.4.)
EquivalentObjectProperties($R_1 \ldots R_n$)	SubObjectPropertyOf(R_1 R_2), SubObjectPropertyOf(R_2 R_1), ...
DisjointObjectProperties($R_1 \ldots R_n$)	DisjointObjectProperties(R_1 R_2), DisjointObjectProperties(R_1 R_3), ...
ObjectPropertyDomain(R C)	SubClassOf(ObjectSomeValuesFrom(R owl:Thing) C)
ObjectPropertyRange(R C)	SubClassOf(ObjectSomeValuesFrom(InverseObjectProperty(R) owl:Thing) C)
InverseObjectProperties(R S)	SubObjectPropertyOf(R InverseObjectProperty(S)), SubObjectPropertyOf(InverseObjectProperty(S) R)
FunctionalObjectProperty(R)	SubClassOf(owl:Thing ObjectMaxCardinality(1 R owl:Thing))
InverseFunctionalObjectProperty(R)	SubClassOf(owl:Thing ObjectMaxCardinality(1 InverseObjectProperty(R) owl:Thing))
ReflexiveObjectProperty(R)	SubClassOf(owl:Thing ObjectExistsSelf(R))
IrreflexiveObjectProperty(R)	SubClassOf(owl:Thing ObjectComplementOf(ObjectExistsSelf(R)))
SymmetricObjectProperty(R)	SubObjectProperty(R InverseObjectProperty(R))
AsymmetricObjectProperty(R)	DisjointObjectProperties(R InverseObjectProperty(R))
TransitiveObjectProperty(R)	SubObjectPropertyOf(SubObjectPropertyChain(R R) R)
ClassAssertion(a C)	SubClassOf(ObjectOneOf(a) C)
ObjectPropertyAssertion(R a b)	ClassAssertion(a ObjectSomeValuesFrom(R ObjectOneOf(b)))
NegativeObjectPropertyAssertion(R a b)	ClassAssertion(a ObjectComplementOf(ObjectSomeValuesFrom(R ObjectOneOf(b))))
SameIndividual($a_1 \ldots a_n$)	ClassAssertion(a_1 ObjectOneOf(a_2)), ...
DifferentIndividuals($a_1 \ldots a_n$)	ClassAssertion(a_1 ObjectComplementOf(ObjectOneOf(a_2))), ...

Table 5.5: Semantics-preserving rewriting of some OWL axioms. C, C_1, ..., C_n and D are class descriptions; R, R_1, ..., R_n and S are property descriptions; and a, b, a_1, ..., a_n are named individuals. Note that in the case of inverting an already inverted property, we simplify the property by $(R^-)^- \mapsto R$.

OWL class	Equivalent OWL class
owl:Nothing	ObjectComplementOf(owl:Thing)
ObjectOneOf($a_1 \ldots a_n$)	ObjectUnionOf(ObjectOneOf(a_1) ... ObjectOneOf(a_n))
ObjectHasValue(R a)	ObjectSomeValuesFrom(R ObjectOneOf(a))
ObjectExistsSelf(InverseObjectProperty(R))	ObjectExistsSelf(R)
ObjectMinCardinality(0 R C)	owl:Thing
ObjectMaxCardinality(0 R C)	ObjectComplementOf(ObjectSomeValuesFrom(R C))
ObjectExactCardinality(0 R C)	ObjectComplementOf(ObjectSomeValuesFrom(R C))

Table 5.6: Semantics-preserving rewriting of some OWL classes. C is a class description, R is a property description, and a, a_1, ..., a_n are named individuals. Note that in the case of inverting an already inverted property, we simplify the property by $(R^-)^- \mapsto R$.

The *SubClassOf*-axioms, however, need further rewriting in order to be compatible with the decisions made in the DCG (e.g. that intersections and unions are always binary and that complex class descriptions are often wrapped into an intersection). The rewriting also performs "sentence planning" to gain better readability of the DCG output, e.g. we reorder elements in coordinations (i.e. the arguments of *ObjectIntersectionOf* and *ObjectUnionOf*) so that simpler noun phrases would occur before complex ones in the verbalization, and avoid negation if possible. Additionally, we must detect structurally complex axioms which the DCG does not support.

Removing class-level syntactic sugar

First, we remove some class-level syntactic sugar from the *SubClassOf*-axioms by rewriting some class descriptions via logically equivalent class descriptions. For example, *ObjectHasValue* can be expressed via *ObjectSomeValuesFrom*, and *ObjectOneOf* containing a list of individuals can be expressed as a union of *ObjectOneOf*-classes containing just one individual, thus making it compatible with the corresponding DCG rule. Cardinality restrictions with 0-cardinality must be rewritten via negation because ACE does not support 0 in numerical quantifiers as it is less natural than negation. Note that in the case of *ObjectMinCardinality(0 R C)* we drop the property and class descriptions which is in slight violation with our design guideline to preserve all the names in the transformation. Still, we think that in this case this violation is acceptable. The complete list of rewriting rules is presented in table 5.6.

Enabling more direct verbalization

Secondly, we wrap every complex class description into *ObjectIntersectionOf* with *owl:Thing* as the first element, in case the complex class description is embedded in a property restriction. I.e. for every $\exists\, R\, C$ in the input axiom, we perform the following

semantics-preserving transformation.

(5.53) $$C \longmapsto (\top \sqcap C)$$

The idea here is to change each class description so that it reflects best the natural syntactic structure. An example of aligning $\exists\, border\ (\exists\, contain\ capital)$ is

$\exists\, border$	$(\top$	$\sqcap$	$\exists\, contain$	$capital)$
borders a	thing	that	contains a	capital

'... borders something that contains a capital'

Reordering *ObjectIntersectionOf* and *ObjectUnionOf*

The order of coordinated classes and positioning of negation is known to contribute to the readability of natural language sentences (in the context of verbalizing OWL, a similar observation is made in [HKKHW05]). For example,

(5.54) Everything that does not own a bike and that is a man and that owns a car ...

is arguably less readable than

(5.55) Every man that owns a car and that does not own a bike ...

In order to provide more readable verbalizations, we reorder classes in coordination so that simple classes come first, i.e. we order lists in *ObjectIntersectionOf* and *ObjectUnionOf* by what we call "natural ordering". We define a "natural ordering" $\prec$ of class description patterns as

(5.56) $$\top \prec C \prec \{a\} \prec R\ \text{SELF} \prec D \prec \neg C \prec \neg\{a\} \prec \neg(R\ \text{SELF}) \prec \neg D \prec E$$

where $x \prec y$ stands for "x comes before y", C stands for a named class, $\{a\}$ for an *ObjectOneOf*-class containing exactly one individual a, R for a property description, D for one of $\exists RC$, $\forall RC$, $\geq nRC$, $\leq nRC$, $= nRC$, and E for all other more complex classes involving embedding or coordination.

Handling *ObjectComplementOf*

We also remove negations as much as possible, again, in order to produce more readable sentences, e.g. the following simplification

(5.57) $$man \sqsubseteq \neg(\leq 5\ own\ book) \longmapsto man \sqsubseteq (\geq 6\ own\ book)$$

corresponds to the ACE-level mapping

(5.58) No man owns at most 5 books. $\longmapsto$ Every man owns at least 6 books.

which arguably improves the readability of the sentence.

In general, there is a way to remove negations from class descriptions and/or push negations inside class descriptions. We can use the following equivalences to push the negation all the way in front of the named class, arriving at the so-called negation normal form [Hor03].

$$\neg\neg C \equiv C \tag{5.59}$$

$$\neg(\leq nRC) \equiv (\geq (n+1)RC) \tag{5.60}$$

$$\neg(\geq nRC) \equiv (\leq (n-1)RC) \tag{5.61}$$

$$\neg\forall RC \equiv \exists R\neg C \tag{5.62}$$

$$\neg\exists RC \equiv \forall R\neg C \tag{5.63}$$

$$\neg(C_1 \sqcap \ldots \sqcap C_n) \equiv \neg C_1 \sqcup \ldots \sqcup \neg C_n \tag{5.64}$$

$$\neg(C_1 \sqcup \ldots \sqcup C_n) \equiv \neg C_1 \sqcap \ldots \sqcap \neg C_n \tag{5.65}$$

Rules 5.59–5.61 that remove negations obviously simplify the corresponding ACE sentences. However, pushing the negation into *ObjectSomeValuesFrom* harms the readability of the eventual verbalization. For example, on the ACE level, keeping the negation in front of a verb ('does not see a dog') and not in front of the noun ('sees something that is not a dog') achieves better readability because in the latter case we would have to introduce 'something' after the verb. The following examples demonstrate the verbalization of *ObjectSomeValuesFrom* and *ObjectAllValuesFrom* in the case of negation, i.e. the ACE-level transformations that correspond to 5.62 and 5.63.

(5.66) something that does not R nothing but C $\longmapsto$ something that R something that is not a C

(5.67) something that does not R a C (best readability) $\longmapsto$ something that R nothing but things that are not a C (worst readability)

We have decided to apply the transformation 5.62, but not the transformation 5.63, see section 5.7.6 for a motivation of this decision.

In the case of coordination, it is hard to decide whether negation should be pushed inside or not, consider the following transformations.

(5.68) something that is not something that is a pianist and that is a violinist (= something that is not a pianist that is a violinist) $\longmapsto$ something that is not a pianist or that is not a violinist

(5.69) something that is not something that is a pianist or that is a violinist $\longmapsto$ something that is not a pianist and that is not a violinist

(5.70) France is not something that is America or that is England or that is Germany or that is Italy. $\longmapsto$ France is something that is not America and that is not England and that is not Germany and that is not Italy.

In the case of coordination, we have decided not to perform these rewritings.

Rejecting certain *SubClassOf*-axioms

ACE cannot express arbitrary embedding of classes, as this would require a grouping construct like parentheses. Therefore, axioms that are too complex are rejected in the verbalization process. We define the notions of "complicated class" and "complicated class-list", and use them to detect structurally complex class patterns and reject the axioms that contain such patterns.

Definition 1 (Complicated class) *A class is (structurally) complicated if and only if it is* ObjectIntersectionOf*,* ObjectUnionOf*, or contains* ObjectIntersectionOf *as an embedded class.*

The following is an exhaustive list of complicated class patterns.

- ObjectIntersectionOf(_)
- ObjectUnionOf(_)
- ObjectComplementOf(ObjectIntersectionOf(_))
- ObjectSomeValuesFrom(_, ObjectIntersectionOf(_))
- ObjectAllValuesFrom(_, ObjectIntersectionOf(_))
- ObjectMinCardinality(_, _, ObjectIntersectionOf(_))
- ObjectMaxCardinality(_, _, ObjectIntersectionOf(_))
- ObjectExactCardinality(_, _, ObjectIntersectionOf(_))

Definition 2 (Complicated class-list) *Complicated class-list is a class-list which contains at least 2 complicated classes.*

An example of a class description which cannot be verbalized in ACE_2 is

$$(\exists R_1\,(\exists R_2\,C_1))\sqcap(\exists R_3\,(\exists R_4\,C_2)) \tag{5.71}$$

because it is first rewritten into

$$(\exists R_1\,(\top\sqcap\exists R_2\,C_1))\sqcap(\exists R_3\,(\top\sqcap\exists R_4\,C_2)) \tag{5.72}$$

which contains a complicated class-list.

Note that the complication for the verbalizer arises from the fact that the verbalizer is bound to use relative clauses which are restricted in how they can attach the subjects and objects of preceding verbs. By using *if-then* sentences in ACE_1, some of those complicated axioms could still be handled (see 5.7.1).

Note that lists that contain just one complicated class could also pose a problem for the verbalization. But natural ordering guarantees that this single complicated class is

placed to the last position where it can be verbalized in an ACE_2-compatible way. For example, without the ordering, the axiom

(5.73) $$man \sqsubseteq \exists\, own\, (dog \sqcap \exists\, hate\, cat) \sqcap \exists\, like\, \{Mary\}$$

is verbalized as

(5.74) Every man is something that owns a dog that hates a cat and that likes Mary.

where the last relative clause 'that likes Mary' would attach to the noun 'dog' according to ACE interpretation rules. This, however, is not the intended reading of the original OWL axiom. Reordering the elements of the conjunction in the OWL axiom, gives the correct verbalization.

(5.75) Every man is something that likes Mary and that owns a dog that hates a cat.

Finally, note that the definition of "complication class-list" only excludes certain complex branching in the class descriptions. Arbitrarily deep embedding is still possible, e.g. ACE_2 sentences can look like the following

(5.76) John does not live-in a country that contains a city that has a mayor that is hated by at least 100 persons that own a bike that is not a mountain-bike.

Completeness

The rewriting of *SubClassOf*-axioms is not complete, i.e. sometimes a class description could be simplified further, e.g. by considering information available in other axioms. For example, a class description $man \sqcup woman$ could be simplified into *person* given that some axiom states that $person \doteq man \sqcup woman$. We have decided not to perform this further simplification. On the one hand, we do not want to employ a full OWL reasoner to help with the rewriting, on the other hand, we want to preserve all the names used in the axioms so that the axiom would not change beyond recognition.

5.6.6 Verbalization algorithm

The verbalization algorithm is simply a sequential application of the methods described above. For each axiom in the OWL ontology, we

1. rewrite it by the mapping described in section 5.6.4;
2. in case the resulting axiom is a *SubClassOf*-axiom, we rewrite it further by the method described in section 5.6.5, otherwise we simply pass it on to the next step;
3. the rewritten axiom is then verbalized by the DCG and the resulting list of ACE tokens is "beautified" and concatenated into a sentence, as described in section 5.6.3.

The result is a list of ACE sentences that follows the order of axioms in the input ontology. Note that there are several further ways to present the ontology, e.g.

- alphabetically;
- in the order: axioms about individuals (i.e. those that have *ObjectOneOf*-description as the first argument of *SubClassOf*), class axioms, property axioms (i.e. *SubObjectPropertyOf* and *DisjointObjectProperties*);
- in the "index view" where all words (OWL names) are listed alphabetically, together with all the sentences that contain them.

The final option is particularly useful in order to get an overview of the usage of the word, i.e. the axioms that contain a certain name. We discuss this presentation option further in section 7.4. Note also that although every OWL serialization (as Functional-Style Syntax, as XML, as RDF triples, etc.) has a certain order, there is no notion of order in the editing model of current OWL editors. Thus the order of resulting ACE sentences might seem random to somebody who has previously looked at the ontology in a graphical OWL editor.

5.7 Discussion

5.7.1 Relationship between ACE_1 and ACE_2

ACE_1 is strictly more expressive than ACE_2, both syntactically and semantically. For example, the following ACE_1 sentence is not expressible in ACE_2 where one is restricted to using *every*-sentences.

(5.77) If a man owns a dog that likes a cat and the man owns a goat that hates a cabbage then the man is a ....

The OWL axiom that corresponds to this sentence

$$(5.78) \qquad \begin{aligned} &man \sqcap (\exists\, own\, (dog \sqcap \exists\, like\, cat)) \sqcap \\ &(\exists\, own\, (goat \sqcap \exists\, hate\, cabbage)) \sqsubseteq \ldots \end{aligned}$$

features a complicated class-list and is thus rejected in the verbalization process. For that reason, the round-trip $ACE_1 \rightarrow$OWL$\rightarrow ACE_2$ is not always defined. However, the violating sentences are so complex that in practice such sentences do not often occur, and even if they do, it could be taken as a signal that the original sentence has poor readability. I.e. the $ACE_1 \rightarrow$OWL$\rightarrow ACE_2$ mapping could be used as a natural test for the complexity/readability of ACE sentences, and/or the OWL$\rightarrow ACE_2$ mapping for the complexity/readability of OWL axioms.

5.7.2 OWL naming conventions

While mapping ACE nouns, verbs and proper names to OWL classes, properties, and individuals, is not problematic, the quality of the OWL→ACE direction (i.e. verbalization) depends on the morphologic and orthographic nature of the names used in the input ontology. This means that probably the most visible deficiency of the described verbalization is caused by the naming conventions used in existing OWL ontologies. The class, property and individual names are not under the control of current OWL editing tools and the user is guided only by informal style-guides, which mainly discuss the capitalization of names (see e.g. [HJM+07, NM01, Goo07]), or are specific to RDF, such as the RoleNoun pattern[4]. Real-world OWL ontologies can contain class names like *FifteenMinutes*, *NAMEDArtery*, *Urgent*, *mirrorImaged*; property names like *hasTopping*, *offeredIn*, *isPairedOrUnpaired*, *accountName*, *brotherOf*, *isWrittenBy*; and individual names like *red*, *married*. Such names do not lend themselves well to any verbalization scheme.

Hopefully, names will become more English-like over time, as ontology languages and tools evolve. For example, OWL 1.1 adds support for anonymous inverse properties (*InverseObjectProperty*) and thus does not force the user to invent a new (and often artificial) name just to be able to talk about an inverse of an existing property. Also, the practice of attaching nouns (i.e. class names) to property names might disappear in the presence of qualified cardinality restrictions (which is another new feature introduced in OWL 1.1).

5.7.3 Deeply nested and branching class descriptions

Looking at some real-world ontologies (e.g. GALEN[5], Ordnance Survey Hydrology ontology[6]), has shown that class descriptions can occasionally be very deep and branch in complex ways. ACE, on the other hand cannot handle certain branching. We see various solutions to this.

First, in an interactive environment, a valid solution would be to reject class descriptions that cannot be verbalized, so that the user could simplify them. We believe that such a natural restriction would result in more readable ontologies in the end.

Secondly, we could add to ACE a support for parentheses. However, this is against the design guidelines of ACE where the formal-looking syntax is preferably avoided.

Third, we could use *if-then* sentences (as done in ACE_1) which are syntactically more expressive than *every*-sentences but lack conciseness.

Fourth, we could try to automatically simplify the ontology by defining new named classes and using them to rewrite the ontology. There are several problems with the last approach — the modified ontology would not be equivalent to the original but would only entail it; the original structure of the class description (which the ontology author

[4] `http://esw.w3.org/topic/RoleNoun`

[5] `http://www.co-ode.org/galen/`

[6] `http://www.ordnancesurvey.co.uk/ontology/`

has maybe carefully constructed) is destroyed; meaningful names are difficult to construct automatically.

Note that we are currently using the first solution, i.e. we reject overly complex class descriptions.

5.7.4 DisjointUnion and other short-hand constructs

OWL 1.1 includes powerful short-hand axioms like *DisjointUnion*, and other forms of syntactic sugar motivated by OWL usage patterns are discussed in the literature [HDG+06, Vra05]. ACE does not provide such short-hands and the verbalization will therefore unravel complex constructions. In some cases, the ACE representation is a long text that is difficult to grasp. The extreme case is

$$DisjointUnion(A\ C_1 \ldots C_n) \tag{5.79}$$

which can be seen as a macro, expanding into two other short-hand axioms

$$EquivalentClasses(A\ ObjectUnionOf(C_1 \ldots C_n)) \tag{5.80}$$

$$DisjointClasses(C_1 \ldots C_n) \tag{5.81}$$

The first of those axioms is not a major problem as it can be expressed in two (possibly long) ACE sentences. Note that we need two sentences as ACE does not provide an *iff*-construct. The *DisjointClasses*-axiom does not have a short corresponding construct in ACE, we need to unravel it into a set of $n*(n-1)/2$ sentences, each in the from "No C_i is a C_j.". As a result, *DisjointUnion*-axioms would be expressed in ACE as (for $n = 3$):

(5.82) No C_1 is a C_2.

(5.83) No C_1 is a C_3.

(5.84) No C_2 is a C_3.

(5.85) Every A is something that is a C_1 or that is a C_2 or that is a C_3.

(5.86) Everything that is a C_1 or that is a C_2 or that is a C_3 is an A.

For instance, *DisjointUnion(person male female)* would be verbalized as

(5.87) No male is a female.

(5.88) Every person is something that is a male or that is a female.

(5.89) Everything that is a male or that is a female is a person.

While this is a valid approach that explains the notion of a covering union of pairwise disjoint classes to a novice OWL user, more experienced OWL users may prefer a more concise verbalization and a way to enter such statements. For example, [RDH+04] recommends tools to provide a wizard-interface to help users enter statements like *DisjointUnion*. This might be the best solution in the end, i.e. it does not make sense to try to come up with a (likely to be complicated) natural language interface in this case. For example, the Sydney OWL Syntax [CSM07] proposes the construct

(5.90) The class person is fully defined as female or male, and female and male are mutually exclusive.

which is hard to read and write, and gets especially awkward if more than two named classes or even complex classes are involved. Furthermore, this construct requires knowledge of notions such as "class" and "mutual exclusion".

5.7.5 Property axioms

The verbalization that we have described can handle most of the OWL 1.1 constructs without using explicit anaphoric references because relative clauses can achieve the same argument sharing effect that anaphoric references would be otherwise needed for. For property axioms (*SubObjectPropertyOf* and *DisjointObjectProperties*), the situation is different. E.g. transitivity of the property 'contain' could be expressed as

(5.91) Everything that contains something that contains something contains it.

where the personal pronoun 'it' refers to the most recent noun phrase (i.e. the last 'something'). Such sentences can be hard to read. Using *if-then* sentences with explicit variables is more clear,

(5.92) If something X contains something that contains something Y then X contains Y.

but still hard to read. On the other hand there does not seem to exist a better natural language representation which would also make explicit the meaning of transitivity.

Note also that the problem with such "fixed" *if-then* sentences is that the users should somehow be discouraged to edit them, because even a minor change, e.g. qualifying the noun phrases or negating the verbs would go beyond OWL's expressivity.

5.7.6 *ObjectAllValuesFrom*

According to [RDH+04], the meaning of *ObjectAllValuesFrom* is often misunderstood by the users. Part of the reason can be that it is hard to come up with an acceptable verbalization for this class description. Such a verbalization must be able to explain that e.g. $(\forall\ eat\ vegetable)$ is to be interpreted as a class of things that either do not eat anything, or if they do then it is only vegetables that they eat. Such an explanation is much

less compact than the original OWL class description. This means that while embedding *ObjectAllValuesFrom* into another class description structure does not increase the overall complexity much, verbalizing the overall structure becomes almost impossible.

Fortunately, *ObjectAllValuesFrom* is not seen as the core element of OWL (as *ObjectSomeValuesFrom* is seen). In order to obtain tractability, many proposed fragments of OWL 1.1 do not support *ObjectAllValuesFrom* at all [Gra07]. Such fragments are *EL++* (used in Gene Ontology[7] and in the medical ontology SNOMED[8]), *DL-Lite*, and *RDFS*.

Our solution is to try to remove *ObjectAllValuesFrom* as much as possible from the input axiom. This works if *ObjectAllValuesFrom* is used together with *ObjectComplementOf*, i.e. we can apply the following rewriting:

$$\neg \forall R\, C \quad \longmapsto \quad \exists R \neg C \tag{5.93}$$

$$\forall R \neg C \quad \longmapsto \quad \neg \exists R\, C \tag{5.94}$$

which preserves the negation but replaces *ObjectAllValuesFrom* with *ObjectSomeValuesFrom*. In addition, simple axiom patterns where *ObjectAllValuesFrom* is used on the right-hand side of the *SubClassOf*-axiom can be rewritten by

$$C \sqsubseteq \forall R\, D \longmapsto \exists R^{-}\, C \sqsubseteq D \tag{5.95}$$

In the general case, however, the only option is to replace *ObjectAllValuesFrom* by double *ObjectComplementOf* which makes the resulting formula structurally too complex, and consequently its verbalization very difficult to understand.

$$\forall R\, C \longmapsto \neg \exists R \neg C \tag{5.96}$$

We have therefore decided to use the determiner 'nothing but' as directly corresponding to $\forall$ (as the determiner 'a' corresponds to $\exists$). This keeps the resulting verbalization as short as for *ObjectSomeValuesFrom*, although the meaning of 'nothing but' is probably more confusing than the meaning of 'a'. Note that other researchers have proposed similar markers, e.g. 'only' (used in [CSM07]), 'only ... or nothing' ([DHK+07]), and 'always' ([HKKHW05]). We find them more ambiguous and misleading than 'nothing but', e.g. 'always' could be interpreted as having a temporal meaning. Also, markers like 'only' can syntactically occur at almost all positions in the sentence, including positions that we semantically do not want to (and cannot) support. We thus need a marker that can syntactically precede nothing but nouns.

[7] http://www.geneontology.org

[8] http://www.snomed.org

Chapter 6

Extensions

6.1 Introduction

In this chapter, we extend the mapping that was discussed in the main part of the thesis. These extensions allow for the use of ACE *of*-constructions to express OWL object properties, thus making some constructs more natural to express (section 6.2). We also discuss a subset of OWL's data property support and how it could be expressed in ACE, again, using *of*-constructions (section 6.3). We then look at two technologies that are often used in conjunction with OWL ontology engineering, namely queries (section 6.4) and rules (section 6.5). Several query and rule formalisms reference OWL concepts, or even overlap with OWL with regard to their semantics (see e.g. [GHVD03]). Yet they use a completely different syntax. It would be desirable to unify the different syntaxes under a single interface. We essentially propose the controlled English approach to achieve this unification.

6.2 *of*-constructions as object properties

6.2.1 Introduction

In this section, we describe an extension that allows us to use the *of*-construction and its syntactic variants Saxon genitive (e.g. 'John's', 'a man's', 'everybody's'), possessive pronoun (e.g. 'its', 'their'), and the relative clause pronoun 'whose', in order to generate OWL's object property restrictions. As all those syntactic variants translate to the DRS condition *relation/3*, we look only at how *relation/3* can be handled. Essentially we will treat it in the same way as the condition *predicate/4* which corresponds to transitive verbs.

The support for *of*-constructions is not motivated by a need for more semantic expressivity but is rather added to align our verbalizations more with the naming conventions in existing OWL ontologies. For example, it is common in current ontologies to use (object) property names like 'part-of' and 'father-of'. For example, suggestions in [NM01] can be summarized as: be consistent, use plural nouns for classes, and use 'has-'

(e.g. ‘has-father’) or ‘-of’ (e.g. ‘father-of’) for slots (i.e. properties). By supporting the *of*-constructions, we enable the user to say

(6.1) Every tissue is a part of an organ.

instead of (or alternatively to)

(6.2) Every tissue is contained by an organ.

6.2.2 Implementation

The DRS representation for sentences like “John’s father is Bill.” uses the DRS-condition *relation/3* which links the object-condition of ‘father’ to the object-condition of the “owner” ‘John’, e.g.

(6.3)

A B C D
object(A, John, named, na, eq, 1) object(B, Bill, named, na, eq, 1) object(C, father, countable, na, eq, 1) predicate(D, be, C, B) relation(C, of, A)

In order to make this DRS acceptable for the DRS→OWL algorithm, and at the same time keep it as a natural representation of the sentence, it suffices to replace the relation-condition with a new predicate-condition that links ‘John’ with ‘father’ and that reuses the name ‘father’ as the name of the predicate.

(6.4)

A B C D E
object(A, John, named, na, eq, 1) object(B, Bill, named, na, eq, 1) object(C, father, countable, na, eq, 1) predicate(D, be, C, B) predicate(E, father, A, C)

This transformation is possible because the relation-condition always occurs in the same DRS box with an object-condition that is identified by the first argument of the relation-condition. Consequently, the OWL representation for this DRS is

```
ClassAssertion(Individual(John) OWLClass(owl:Thing))
ClassAssertion(Individual(Bill) OWLClass(owl:Thing))
ClassAssertion(AnonymousIndividual(1) OWLClass(father))
SameIndividual(AnonymousIndividual(1) Individual(Bill))
ObjectPropertyAssertion(ObjectProperty(father) Individual(John)
                                               AnonymousIndividual(1))
```

Note that this transformation profits from the punning-feature of OWL 1.1 as the name ‘father’ acts as both a class name and an individual name. As on the ACE level, the noun ‘father’ is accessible for anaphoric references, we need to keep it available also on the OWL level, by introducing an anonymous individual of class ‘father’, i.e. one can write

(6.5) John's father is Bill. The father likes Mary.

to introduce a property assertion between the unnamed father and Mary.

```
ObjectPropertyAssertion(ObjectProperty(like) AnonymousIndividual(1)
                                             Individual(Mary))
```

6.3 Data properties

6.3.1 Introduction

Like many other knowledge representation languages, due to practical requirements, OWL supports data properties such as weight, name, or age that take values of type integer, real or string, from the domain with built-in predicates, such as $\leq$ or $=$. For example, the following axiom describes those humans whose age is 17 as minors.

(6.6) $$\mathit{human} \sqcap \exists \mathit{age}\, 17 \sqsubseteq \mathit{minor}$$

We have added support for data properties in ACE, e.g. one can assert that

(6.7) Every human whose age is 17 is a minor.

where the number 17 is used in the noun phrase position.

In this section, we study the support that OWL 1.1 provides for talking about data properties and concrete domains, and how ACE matches this support. Section 6.3.2 discusses how so-called data ranges can be defined on the basis of existing data values or predefined datatypes such as integer and string. Section 6.3.3 discusses how (object) classes can be defined via data properties and data ranges. Section 6.3.4 lists the axioms that can be used to place constraints of data properties or assert information about the connection of individuals to data values. Section 6.3.5 discusses the corresponding ACE constructs.

6.3.2 Data ranges

OWL 1.1 provides several ways to define a range over data values. A datatype is a fundamental type of data range that is defined by a URI. The list of datatypes supported in OWL 1.1 (*string*, *boolean*, *nonNegativeInteger*, ...) can be extended by implementations as needed [GM07]. Each datatype URI is associated with a predefined arity. E.g. the arity of $\leq_{17}$ is one, while the arity of $\leq$ is two, and the arity of $+$ is three. Complex data ranges can be constructed from the simpler ones using the

- *DataComplementOf*(DR) construct, which takes a data range *DR* and returns its complement (with the same arity);
- *DataOneOf*$(v_1 \ldots v_n)$ construct, which specifies a set of constants $\{v_1^D, \ldots, v_n^D\}$ (and it has arity one);

- *DatatypeRestriction*(*DR f v*) construct which creates a data range by applying a facet *f* with argument *v* to a given data range *DR*, i.e. a restriction consists of a constant restriction value (e.g. *12*) and a facet type (e.g. *minExclusive*) that is applied to the data range in question (e.g. *integer*). OWL 1.1 supports the following facet types: *length*, *minLength*, *maxLength*, *pattern*, *minInclusive*, *minExclusive*, *maxInclusive*, *maxExclusive*, *totalDigits*, *fractionDigits*. The semantics of the facets is defined in [BM04]. E.g. *minExclusive* is defined as the exclusive lower bound of the value space for a datatype with the "ordered" property.

Example 1 *Range of non-negative integers that are larger than 12.*

DatatypeRestriction(http://www.w3.org/2001/XMLSchema#nonNegativeInteger minExclusive 12)

6.3.3 Classes defined via data properties and data ranges

As is the case with object properties, also data properties can be used to define classes. For example, the class of teenagers can be defined as "every person who has an age that is from the data range 13–19". OWL 1.1 supports the following class descriptions based on data properties and ranges:

- *DataSomeValuesFrom*($U_1 \ldots U_n$ *DR*),
- *DataAllValuesFrom*($U_1 \ldots U_n$ *DR*),
- *DataHasValue*(*U v*),
- *DataMinCardinality*(*n U DR*),
- *DataMaxCardinality*(*n U DR*),
- *DataExactCardinality*(*n U DR*);

where U, U_1, ..., U_n are data properties, *DR* is a data range, v is a data value, and n is a non-negative integer.

In the following, we discuss the interpretation of *DataSomeValuesFrom* and *DataHasValue*, and refer to [GM07] for the interpretation of the rest of the class descriptions.

DataSomeValuesFrom($U_1 \ldots U_n$ *DR*) is interpreted as a set of objects

$$\{x \mid \exists y_1, \ldots, y_n : (x, y_k) \in U_k^I \text{ for each } 1 \leq k \leq n \text{ and } (y_1, \ldots, y_n) \in DR^D\} \tag{6.8}$$

where U^I is data property interpretation function that assigns to each data property a subset of $\Delta_I \times \Delta_D$ (i.e. pairs of elements taken from the object domain and data domain), DR^D assigns to each data range with arity n an n-ary relation over Δ_D. Note that the *DataSomeValuesFrom*-restriction takes a **list** of data properties, and not just a single property expression, as an argument. This is in order to support class definitions such as

"objects whose width is greater than their height", where the values of width and height are specified using two data properties. In such definitions, the arity of the given data range must be equal to the number of the given data properties.

Example 2 *To express the class of things whose width is greater than their height.*

DataSomeValuesFrom(width height http://ranges.net/greater)

$DataHasValue(U\ v)$ is interpreted as

$$\{x \mid (x, v^D) \in U^I\} \tag{6.9}$$

and is thus a specific form of *DataSomeValuesFrom*, i.e.

$$DataHasValue(U\ v) \equiv DataSomeValuesFrom(U\ DataOneOf(v)) \tag{6.10}$$

6.3.4 Data property axioms

Data properties can have a smaller number of properties than object properties in OWL 1.1, because we cannot talk about transitivity, inverse properties and symmetric properties, as the domain and the range of data properties come from disjoint sets (objects vs. data values). In addition, OWL does not allow data properties to be defined to be inverse functional. So, we are left with defining subproperties, domains, ranges, functionality, disjointness, and asserting data properties between individuals and data values. OWL 1.1 supports the following data property axioms:

- $SubDataPropertyOf(U\ V)$,
- $EquivalentDataProperties(U_1 \ldots U_n)$,
- $DisjointDataProperties(U_1 \ldots U_n)$,
- $DataPropertyDomain(U\ C)$,
- $DataPropertyRange(U\ D)$,
- $FunctionalDataProperty(U)$,
- $DataPropertyAssertion(U\ a\ v)$,
- $NegativeDataPropertyAssertion(U\ a\ v)$;

where $U, V, U_1, \ldots, U_n$ are data properties, C is a class, a is an individual, v is a data value, and DR is a data range with arity one.

Again, we refer to [GM07] for the interpretation of those axioms and note here only that the assertion-axioms can be rewritten via already introduced OWL constructs, namely by

$$
\begin{aligned}
\mathit{DataPropertyAssertion}(U\ a\ v) &\equiv \mathit{SubClassOf}(\mathit{ObjectOneOf}(a) \\
&\quad\ \mathit{DataHasValue}(U\ v)) \\
\mathit{NegativeDataPropertyAssertion}(U\ a\ v) &\equiv \mathit{SubClassOf}(\mathit{ObjectOneOf}(a) \\
&\quad\ \mathit{ObjectComplementOf}(\mathit{DataHasValue}(U\ v)))
\end{aligned}
$$

For example, in order to express that John's age is 18, we could write

(6.11) DataPropertyAssertion(age John 18)

Note that in this example, we should probably add that 'age' is functional, i.e. that nobody can have more than one age. Also, we could assert that the domain of 'age' contains only living beings and the range only positive numbers. Such constraints can be used to type-check all further assertions.

6.3.5 Expressing data properties in ACE

Introduction

In OWL, data properties connect syntactically a class or an individual to a data value or range. Similarly, in ACE, if the object of the verb or verb-like construct is a number or a string, then the verb could be taken to correspond to a data property.

As data property names, we have decided to used both regular transitive verbs (e.g. "There is a thermometer that **approaches** -30."), and also the genitive expressed by *of*-constructs ("the **value of** the thermometer is -30"), Saxon genitive ("the thermometer**'s value** is -30"), or by relative clauses ("a thermometer **whose value** is -30", "a thermometer the **value of which** is -30"). As existing ontologies mostly use nouns (e.g. 'age', 'weight') for datatype properties, the following examples and discussion focus on the *of*-construction, although regular transitive verbs would provide more syntactic flexibility.

Datatypes, data values, and data ranges

ACE has support for three datatypes: string, integer, and real. Examples of these types of data values are: "*Stop*!", 12, 17.71. They are all morphologically distinguishable and need therefore no explicit typing on the surface level of an ACE text. An ACE parser is expected to automatically derive the datatype of a data value by its lexical properties, i.e. it applies a function φ which returns the type

- *integer*, if the value contains only digits;
- *real*, if the value contains only digits apart from a single dot;
- *string*, if the value is in quotation marks.

ACE	OWL
at least v	DatatypeRestriction($\varphi(v)$ minInclusive v)
at most v	DatatypeRestriction($\varphi(v)$ maxInclusive v)
more than v	DatatypeRestriction($\varphi(v)$ minExclusive v)
less than v	DatatypeRestriction($\varphi(v)$ maxExclusive v)
`not` *ACEDR*	DataComplementOf(ψ(*ACEDR*))
v_1`,` v_2`, ..., or` v_n	DataOneOf($v_1 \ldots v_n$)

Table 6.1: Mapping ψ of ACE data ranges (*ACEDR*) to a subset of OWL data ranges, where v, $v_1, \ldots v_n$ are ACE data values, the main data range on which the facet is applied is derived from the type of an ACE data value by function φ which returns an XML Schema datatype that corresponds to *integer*, *real*, or *string*.

In the context of OWL, these datatypes correspond to the XML Schema's built-in types — *string* corresponds to XML Schema's *string*, *integer* corresponds to XML Schema's *integer*, and *real* corresponds to XML Schema's *double*.

We extend the ACE syntax to support some OWL data ranges and facets by allowing operators listed in table 6.1.

Using data properties in ACE

The usage of data properties in ACE maps to OWL axioms which only use a restricted (unary) form of *DataSomeValuesFrom*, namely

$$DataSomeValuesFrom(U\ \psi(ACEDR)) \tag{6.12}$$

where $\psi(ACEDR)$ is a data range that can be derived from ACE data ranges. This lets us express frequently occurring class descriptions via noun phrases like

(6.13) somebody whose age is more than 18

(6.14) somebody whose address is not "Paris"

Data property assertions (i.e. *DataPropertyAssertion* and *NegativeDataPropertyAssertion*) could be made via fairly natural sentences like

(6.15) John's age is at least 12.

(6.16) John's age is not 20.

(6.17) John's age is 11, 12, or 13.

General *SubClassOf*-axioms can be expressed as usual, by *every*- and *if-then* sentences. For example, the sentence

(6.18) If a person's name is "John" and he likes Google then the person's email-address is "john.*@gmail.com".

translates to the OWL axiom

```
SubClassOf(
   ObjectIntersectionOf(
      OWLClass(person)
      DataHasValue(DataProperty(name) "John"^^xsd:string)
      ObjectSomeValuesFrom(ObjectProperty(like) ObjectOneOf(Individual(Google))
      )
   )
   DataHasValue(DataProperty(email-address) "john.*@gmail.com"^^xsd:string)
)
```

6.3.6 Conclusion

We conclude that some OWL 1.1 data property constructs and data ranges could be expressed in a fairly natural way. However, this is not the case for all constructs and range types. Also, in the case of OWL class descriptions and axioms, we have decided to support only the most natural (and probably most frequent) constructs — we have added support for a restricted *DataSomeValuesFrom* (with *DataHasValue* as its special case), and axioms *DataPropertyAssertion*, and *NegativeDataPropertyAssertion* which both can be rewritten as *SubClassOf*-axioms that make use of *DataHasValue*.

6.4 Queries

6.4.1 Introduction

Given a language for reading and writing ontologies, it is also desirable to use this language to query ontologies. It would be a shortcoming if a completely new language was needed to formulate queries over ontologies. It is thus important to extend the discussed ACE fragments to also support queries. An obvious extension is the addition of query words ('who', 'what'). And this is essentially the only addition, i.e. the user must only learn a few new function words, otherwise the language is already familiar. For example, the user already knows that the following questions would ask for the same information, simply because replacing the query pronoun 'who' with the same noun phrase in both questions (and replacing the question mark with the period) would result in semantically equivalent declarative sentences.

(6.19) Who is a man that John sees?

(6.20) Who that John sees is a man?

Also, the DRS representation of those ACE questions is similar to the representation of the corresponding ACE declarative sentences. The only difference is the use of the DRS-condition *query* instead of *object* to denote the query pronoun 'who'.

(6.21)

A B C D E
object(A, John, named, na, eq, 1) query(B, who) object(C, man, countable, na, eq, 1) predicate(D, see, A, C) predicate(E, be, B, C)

Natural language based query-interfaces to databases (and knowledge bases) have been a research topic for many years. Some of the recent work that targets those issues and uses ACE is e.g. [BKFvB04, FS03]. [BKFvB04] converts Attempto DRSs into process query language (PQL). As a result, ACE sentences are mapped into a representation that can be used to directly execute queries against an existing database. The results of the query, however, are presented rather formally, in a table of subject-predicate-object triples. The ACE reasoner RACE [FS03] allows to query an ACE text with ACE queries. The query result is a listing of ACE sentences (a subset of the ACE text) on the basis of which the input query can be answered.

Many different query languages have been proposed for the Semantic Web, one of which, the RDF query language SPARQL [PS07], has reached the stage of a W3C Candidate Recommendation. A SPARQL query is a partially instantiated RDF graph (a set of RDF triples), and the query is evaluated by matching the query graph against the RDF knowledge base, in order to obtain the variable bindings. As such, SPARQL is not directly usable with OWL ontologies. For that reason, several native OWL query languages have been proposed over the recent years, e.g. OWL-QL [FHH03], nRQL [HMW04], OWL SAIQL [KSSP07], SPARQL-DL [SP07]. None of them, however, has been standardized and entered the mainstream.

In this section, we look at two general query frameworks that can be used with OWL — DL Queries (subsection 6.4.2) and conjunctive queries (subsection 6.4.3) — and how they can be expressed in ACE. We do not discuss how the query results could be presented.

6.4.2 DL Queries

A DL Query (as implemented e.g. in the DL Query Tab Widget[1] for Protégé 4) is expressed as a description logic class description, and asks for all the named individuals or classes that are related to the queried class description. For example, one can ask for individuals that belong to the class denoted by the query, or ask for classes that are sub classes, super classes or equivalent to the query.

Supporting DL Queries in ACE is quite straight-forward — the DRS corresponding to an ACE query is to be treated in the same way as embedded DRSs are treated by the DRS→OWL algorithm (described in section 5.5.6), i.e. as a directed graph with exactly one distinguished node. In the case of DL Query, the distinguished node corresponds to the query variable (which in turn corresponds to the ACE query pronoun). Consequently, only those ACE queries that contain exactly one query pronoun are supported as DL Queries.

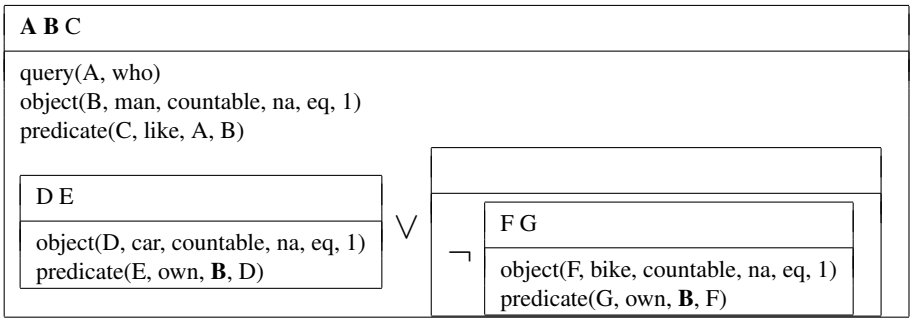
[1] `http://protegewiki.stanford.edu/index.php/DL_Query`

The DRS→OWL algorithm performs the rolling up procedure on the DRS that corresponds to the input ACE query. If the rolling up succeeds then a class description that constitutes the query is formed. For example, the ACE query

(6.22) Who likes a man that owns a car or that does not own a bike?

corresponds to the following DRS

(6.23)



Starting the rolling up from the discourse referent 'A' and consuming all the DRS conditions, results in the class description

(6.24) $$\exists\, like\ (man \sqcap ((\exists\, own\ car) \sqcup (\neg \exists\, own\ bike)))$$

6.4.3 Conjunctive queries

Conjunctive queries (CQs) (see e.g. [HT00]) correspond to the fragment of first-order logic whose formulas are conjunctions of atoms over constants, existentially quantified variables that may be shared across the atoms, and free variables (also called distinguished variables). For example, in the query

(6.25) $$\text{is-parent-of}(z, Bill) \wedge \text{is-parent-of}(z, ?x) \wedge \text{is-parent-of}(z, ?y) \wedge hates(?x, ?y)$$

the atoms are *is-parent-of* and *hates*, the constants include only *Bill*, the existentially quantified variables include only z, and the distinguished variables to be bound are x and y. This query asks for all pairs of Bill's siblings where one of the siblings hates the other one.

Thus, a conjunctive query corresponds to a DRS with no embedded DRS-boxes, where the referents of named-object conditions correspond to constants, query referents to distinguished variables, and all other referents to existentially quantified variables. The *object* and *predicate* conditions map to CQ atoms which are conjoined as are the conditions in the top-level DRS.

CQs allow any variable patterns in the DRS conditions, also those that cannot be rolled up. Also, the corresponding DRSs can have any number of *query/2*-conditions. In the case of no query-conditions, we can talk about a boolean query with an answer "yes" or "no", e.g.

(6.26) Does John like Mary?

and in the case of multiple query-conditions, the answer would bind the query variables to existing individuals. For example, in the DRS-representation of the question

(6.27) Who visits what?

the discourse referents *Ref1* and *Ref2* in *query(Ref1, who)* and *query(Ref2, what)* have to be bound.

6.5 Rules

6.5.1 Introduction

Over the recent years, researchers have come up with a number of rule formalisms and rule syntaxes that fit into the Semantic Web framework. Some of those formalisms are e.g. RuleML[2], Web Rule Language (WRL)[3], Semantic Web Services Language (SWSL)[4], The Web Service Modeling Language WSML[5], Semantic Web Rule Language First-Order Logic (SWRL FOL)[6], REWERSE Rule Markup Language (R2ML)[7]. ACE has previously been used as a rule language in [Kuh07] where a rule system AceRules is developed. AceRules can "execute" ACE texts by one of three different forward-chaining interpreters, thus giving the text three different semantics. The output of the interpreters, i.e. the answer set, is also presented in ACE.

In the following, we look only at the Semantic Web Rule Language (SWRL) [HPSB+04] because it is a natural extension of OWL, and is furthermore widely used because support for it is included in the Protégé ontology editor, in the form of SWRLTab[8].

6.5.2 Semantic Web Rule Language (SWRL)

SWRL [HPSB+04] extends the set of OWL axioms to include Horn-like rules. Those rules have the form of an implication between an antecedent (a body) and consequent (a head). Both the body and the head consist of positive conjunctions of atoms, i.e. SWRL does not support negated or disjoined atoms. Atoms in these rules have the form $C(x)$, $P(x,y)$, $sameAs(x,y)$, $differentFrom(x,y)$, or $builtIn(r,x,\ldots)$, where C is an OWL class description (either named or complex), P is an OWL property name, r is a built-in relation, and x and y are either variables, OWL individuals, or OWL data values. For example, the following is a SWRL rule.

[2] http://www.ruleml.org/
[3] http://www.w3.org/Submission/2005/08/
[4] http://www.daml.org/services/swsf/1.0/swsl/
[5] http://www.wsmo.org/wsml/wsml-syntax
[6] http://www.w3.org/Submission/2005/01/
[7] http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/6
[8] http://protege.cim3.net/cgi-bin/wiki.pl?SWRLTab

$$publication(p) \wedge author(y,p) \wedge author(z,p) \longrightarrow collaborate\text{-}with(y,z) \tag{6.28}$$

SWRL rules are interpreted as: if the atoms specified in the body hold, then the conditions specified in the head must also hold. An atom $C(x)$ holds if x is an instance of the class description, an atom $P(x,y)$ holds if x is related to y by property P, an atom $sameAs(x,y)$ holds if x is interpreted as the same object as y, an atom $differentFrom(x,y)$ holds if x and y are interpreted as different objects, and $builtIn(r,x,\ldots)$ holds if the built-in relation r holds on the interpretations of the arguments.

The power of SWRL lies in its ability to express arbitrary variable sharing and in its support for (possibly user-defined) built-ins. A built-in is a predicate that takes one or more arguments and evaluates to true if the arguments satisfy the predicate. For example, an *equal* built-in can be defined to accept two arguments and return true if the arguments are the same. [HPSB$^+$04] specifies a number of core built-ins for common mathematical operations (e.g. *greaterThan*, *multiply*, *sin*), string operations (e.g. *substring*, *upperCase*), date and time operations (e.g. *subtractDates*), and list operations (e.g. *member*, *listConcat*, *first*). Most of the built-ins have a counterpart in XQuery and XPath languages [MMW07]. List operators have been inspired by similar operators in Prolog and Lisp.

6.5.3 Expressing SWRL in ACE

Mapping DRSs to SWRL rules is trivial, given that we do not consider data values, data properties and built-ins.

The DRS→SWRL mapping can be applied to DRSs where the top-level DRS contains only object-conditions (those map to OWL individuals) and implication-conditions (those map to SWRL rules, where the left-side implication box maps to the SWRL body and the right-side to the SWRL head).

In every implication-condition, every object-condition maps to SWRL named class atom, every predicate-condition maps to SWRL named property atom, every copula-predicate-condition maps to SWRL *sameAs*-atom, every negated copula-predicate-condition maps to SWRL *differentFrom*-atom, i.e.

$$object(X,\text{something},\text{dom},\text{na},\text{na},\text{na}) \longmapsto owl{:}Thing(X) \tag{6.29}$$

$$object(X,\text{somebody},\text{countable},\text{na},\text{eq},1) \longmapsto owl{:}Thing(X) \tag{6.30}$$

$$object(X,\text{thing},\text{countable},\text{na},\text{eq},1) \longmapsto owl{:}Thing(X) \tag{6.31}$$

$$object(X,Name,\text{countable},\text{na},\text{eq},1) \longmapsto Name(X) \quad \text{if } Name \text{ is not 'thing', 'something', } \ldots \tag{6.32}$$

$$predicate(\_,\text{be},X,Y) \longmapsto sameAs(X,Y) \tag{6.33}$$

$$predicate(\_,Name,X,Y) \longmapsto Name(X,Y) \tag{6.34}$$

(6.35) if *Name* is not 'be'

(6.36) $$\neg[predicate(\_, \text{be}, X, Y)] \longmapsto differentFrom(X, Y)$$

where the discourse referents X and Y are mapped either to SWRL variables, or individuals. The latter mapping is applied in case the discourse referent is declared on the top-level, i.e. it corresponds to a proper name or a top-level noun.

Negation and disjunction conditions can be mapped to SWRL atoms if their condition lists (CL, CL_1, CL_2) can be rolled up into OWL class descriptions (C, C_1, C_2).

(6.37) $$\neg CL \longmapsto (\neg C)(X) \text{ where } X \text{ is distinguished in } CL$$

(6.38) $$CL_1 \vee CL_2 \longmapsto (C_1 \sqcup C_2)(X) \text{ where } X \text{ is distinguished in } CL_1 \text{ and } CL_2$$

Note that this mapping can be easily reversed, thus allowing for verbalization of SWRL rules via existing DRS verbalizers.

Chapter 7

Implementation

7.1 Introduction

Figure 7.1 shows a diagram of the various translators implemented by the author of this thesis. Also, the already existing translators that convert ACE texts to DRSs (ACE parser) and DRSs to ACE texts (DRS verbalizer) are shown.

The essential translators for this thesis map the Attempto DRS (of an ACE_1 text) into OWL 1.1 Functional-Style Syntax, SWRL functional syntax, and into DL Query expressed as an OWL class description in the OWL 1.1 Functional-Style Syntax. In all cases, the output is in Prolog-compatible notation (using Prolog lists to denote lists and sets, quoting capitalized atom names, etc.). A further translator maps the combination of OWL+SWRL into RDF so that it can be loaded by OWL and SWRL tools. The verbalization of OWL ontologies is done by using the OWL 1.1 XML syntax as input. The XML notation is first converted into the more convenient Prolog notation (that expresses the OWL 1.1 Functional-Style Syntax), and then verbalized in ACE_2. The ACE View plug-in for Protégé 4 generates OWL 1.1 XML to be converted into ACE, or expects RDF/XML (expressing OWL 1.1 and SWRL) to be merged with Protégé's internal representation of the current ontology.

In the following sections, we describe the ACE→OWL/SWRL translator (section 7.2), the OWL verbalizer (section 7.3), and an end-user tool "ACE View" which uses these two translators as its main components (section 7.4).

7.2 ACE→OWL/SWRL translator

The ACE→OWL/SWRL translator uses the ACE Parsing Engine (APE) to transform an ACE text into its corresponding DRS. If this transformation is successful (i.e. the ACE syntax was not violated), then the translator attempts to convert the DRS into OWL (as described in 5.5.6, but with addition of the support for *of*-constructions as described in 6.2 and for data properties as described in 6.3.5). In case an implication-condition fails to translate, then an attempt is made to convert the implication into SWRL (as described in 6.5.2). If this fails as well, then an error message is returned. In the case of

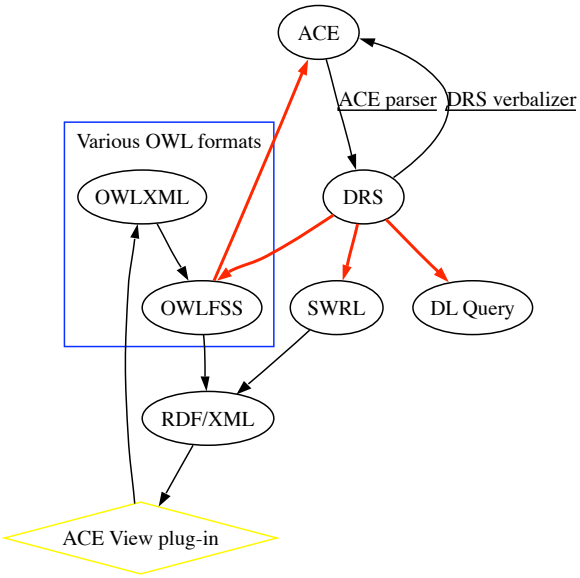


Figure 7.1: Inputs and outputs of the implemented translators. On this diagram, nodes are languages and arrows are translators between the languages. Essential translators that are described in this thesis are marked by bold arrows. Three of those translators take a DRS as input and convert it into OWL 1.1 Functional-Style Syntax (OWLFSS), SWRL, or DL Query, respectively. Another translator takes OWLFSS as input and verbalizes it in ACE.

success, the output is either a pure OWL ontology, or an ontology that mixes OWL and SWRL. The native output of ACE→OWL/SWRL is in functional notation, but optionally an RDF/XML representation can be produced. This is needed to be compatible with existing OWL and SWRL tools, as they generally support only RDF-based syntaxes. The translator is publicly available as a part of the APE web-service[1] and web-client[2]. The translator is implemented in SWI-Prolog.

7.3 OWL verbalizer

The OWL verbalizer accepts an OWL ontology in OWL 1.1 XML syntax as input. The XML representation is first converted by a straight-forward transformation into OWL 1.1 Functional-Style Syntax in Prolog notation. The resulting ontology is then verbalized in ACE as described in section 5.6.6. The OWL verbalizer is implemented in SWI-Prolog.

[1] http://attempto.ifi.uzh.ch/site/docs/ape_webservice.html
[2] http://attempto.ifi.uzh.ch/ape/

7.4 ACE View plug-in for Protégé OWL editor

7.4.1 Introduction

In order to be able to compare our approach to existing ontology engineering approaches, we have integrated the ACE→OWL and OWL→ACE mappings into the widely-used Protégé OWL editor [HJM+07]. This integration is realized as a Protégé plug-in called "ACE View". Specifically, we use the (currently alpha version) Protégé 4[3].

Using Protégé 4 and its underlying OWL API [HBN07] as a platform, gives us access to OWL reasoners Pellet [SPG+07] and FaCT++ [TH06] which can be used to check the consistency of the ontology, entail new axioms on the basis of asserted axioms, and answer DL Queries. The OWL API also supports the explanation of entailments via a Black Box OWL Debugger [HBN07]. Along with OWL axioms, also SWRL rules can be stored and manipulated using the OWL API.

Using an existing ontology editor as a host environment alleviates some of the problems discussed in section 5.7. For example, the host environment can take care of things that are easier to handle by forms (such as entering data about individuals) and wizards (e.g. entering *DisjointUnion*), and support maintaining a lexicon with all the required linguistic information used in the verbalization (e.g. forms of morphologically exceptional words). The users can thus profit from the synergy resulting from the combination of traditional form-based ontology editing and natural language-based editing.

7.4.2 Protégé and ACE View

The standard "Protégé view" to an OWL ontology involves tabs for classes, properties and individuals. Each of those tabs contains several sub windows, e.g. a display of the tree hierarchy of *SubClassOf*-relationships between named classes, a listing of individuals, lists of complex class descriptions (rendered in Manchester Syntax [HDG+06]). Protégé also provides several of the so-called "Ontology views", most of which show the various OWL representations (RDF/XML, OWL 1.1 XML, etc.) of the whole ontology, or general metrics of the ontology (DL expressivity, counts of various OWL constructs).

The "ACE View" provides an alternative "Ontology view", where the complete logical content of the ontology is shown in ACE. In this rendering, ACE sentences correspond to OWL axioms, and all metrics are linguistic, e.g. number of sentences and content words in the ACE text. The ACE view can be edited — sentences can be modified and deleted, and new sentences can be added. A single "Synchronize" button is currently provided to let the user trigger the synchronization of the edited ACE representation with the underlying Protégé representation of the ontology. (In the future, we will try to make the synchronization fully automatic.) A "Preferences" button allows the user to configure the web-services that provide the ACE→OWL and OWL→ACE translators. In addition to those two buttons, six tabs present different views to the ACE text, and thus to the whole ontology.

[3] http://www.co-ode.org/downloads/protege-x/

1. The “Main tab” provides a plain text ACE-representation of the complete ontology, allowing the user to modify the text via standard editing commands such as copy and paste.

2. The “Index tab” provides a more structured representation of the text, using HTML for rendering and navigation.

3. The “Paraphrase tab” provides a paraphrase of the ACE text.

4. The “Inferences tab” shows the ACE representation of the axioms that the ontology entails together with their explanation.

5. The “Answers tab” lets the user query the knowledge base using ACE questions. The answers are given as lists of ACE words or sentences using those words.

6. The “Debug tab” gives a list of all entered ACE sentences along with some technical details about the parsing results for those sentences. In the case of parsing failure, error messages are reported that help the user to rephrase the sentence in an ACE-compatible way.

The following sections describe the tabs in more detail and show screenshots of our current implementation. Note that not all aspects of these tabs are fully implemented at the moment. Therefore, we expect some changes to occur in the design and function of these tabs.

7.4.3 Main

In the main tab (figure 7.2), the current ACE text is displayed and can be edited. Pressing the “Sync” button, triggers the updates to the text to be parsed and integrated into the ontology. Although the user is expected to enter sentences which can be mapped to OWL, inputing sentences that are not ACE or that map only to SWRL is tolerated. Such sentences, however, do not participate in reasoning. The “Debug tab” provides explanations of why a certain sentence could not be parsed. Such a sentence can be modified at any time to comply with ACE, or it can be left around as a “comment”.

7.4.4 Index

In the “Index tab” (figure 7.3), the complete ACE text is presented as an index — the set of content words is alphabetically sorted and every content word is listed together with all the sentences that contain the word. Every content word in a sentence is furthermore a hyper-link to the entry of the content word, thus allowing for easy navigation in the index.

Every sentence that was not successfully parsed into OWL or SWRL is marked by a red label `/*not OWL nor SWRL*/`. Every sentence that was not successfully parsed into OWL but that could be parsed into SWRL is labeled as `/*SWRL*/`.

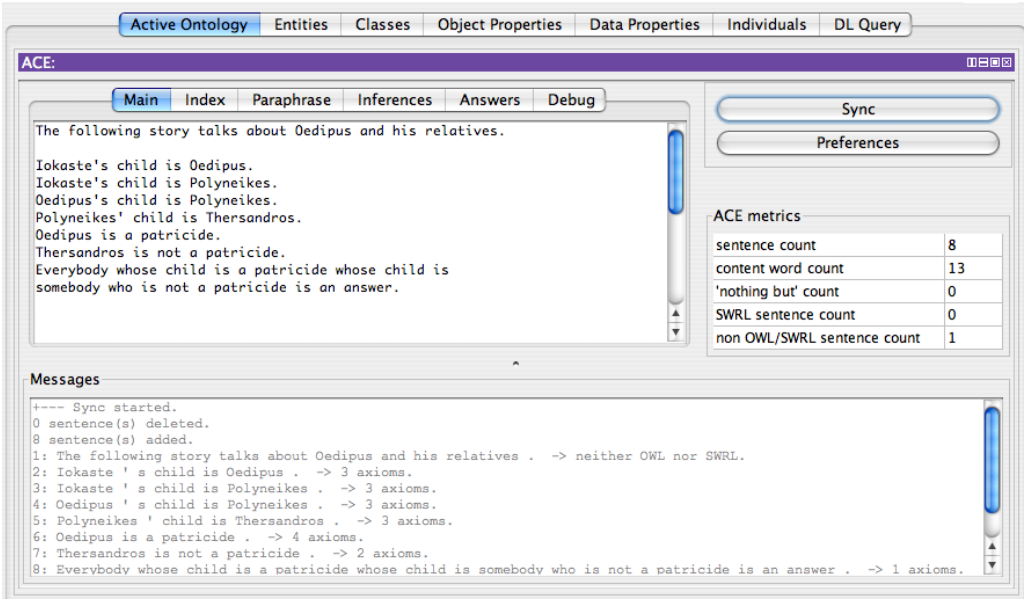


Figure 7.2: Screenshot of the "Main tab" of the ACE View plug-in for Protégé 4. The complete ontology is displayed in one text area.

7.4.5 Paraphrase

In the "Paraphrase tab", a paraphrase of the ACE text is provided. A paraphrase is one way for the user to check if his/her interpretation of the inserted text is accurate. ACE provides many forms of syntactic sugar, thus allowing for paraphrasing, e.g. *every*-sentences can be rephrased via *if-then* sentences, and in many cases vice-versa. At the moment, the paraphrase is a verbalization of the whole ontology via the OWL→ACE mapping. In the future, we will allow the user to select other forms of paraphrasing, e.g. the ones based on Core ACE [FKS05] or NP ACE [FKK06a].

7.4.6 Inferences

The "Inferences tab" (figure 7.4) provides a list of ACE sentences that correspond to the entailed axioms of the ontology. Such axioms can be automatically generated by the built-in reasoner. These axioms have a very simple structure, i.e. they are class assertions, property assertions and sub class axioms where the involved individuals, properties, and classes are always named. Thus their natural language verbalization cannot potentially bring significant usability improvement. Nevertheless, the presentation of all entailments as a single list of natural language sentences can provide a good and easily readable overview. Alternatively, an index view to the entailments could be provided.

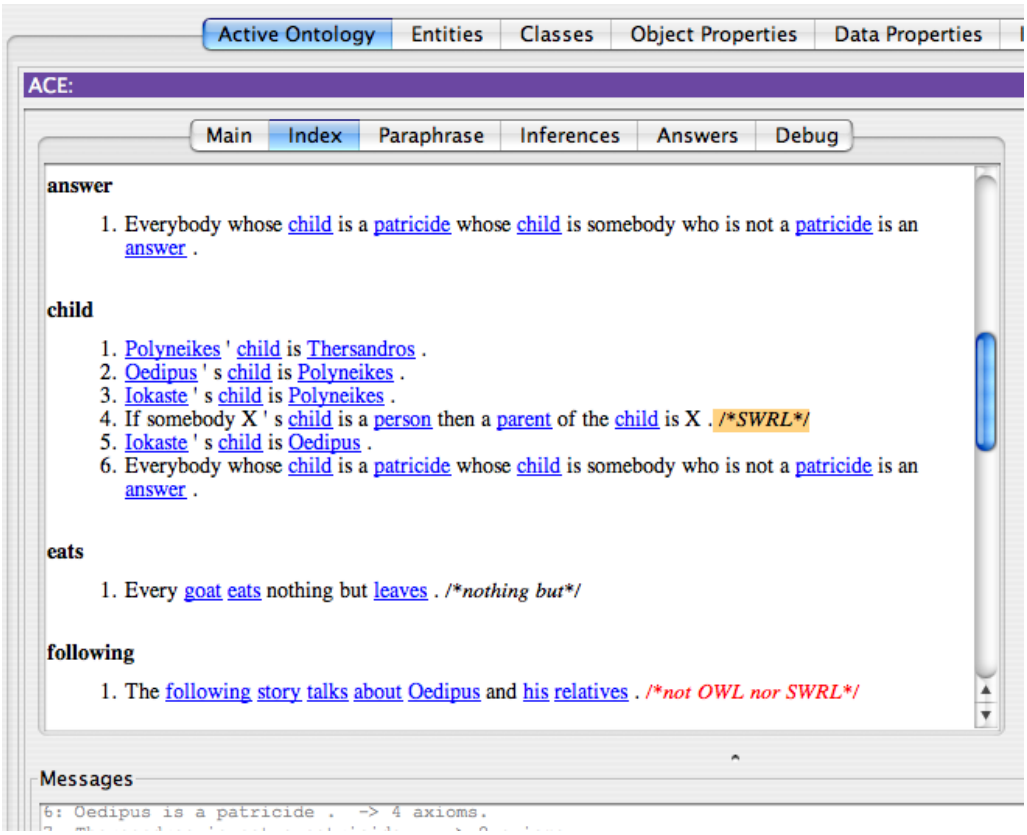


Figure 7.3: Screenshot of the "Index tab" of the ACE View plug-in for Protégé 4. The complete ACE text is indexed and rendered in HTML.

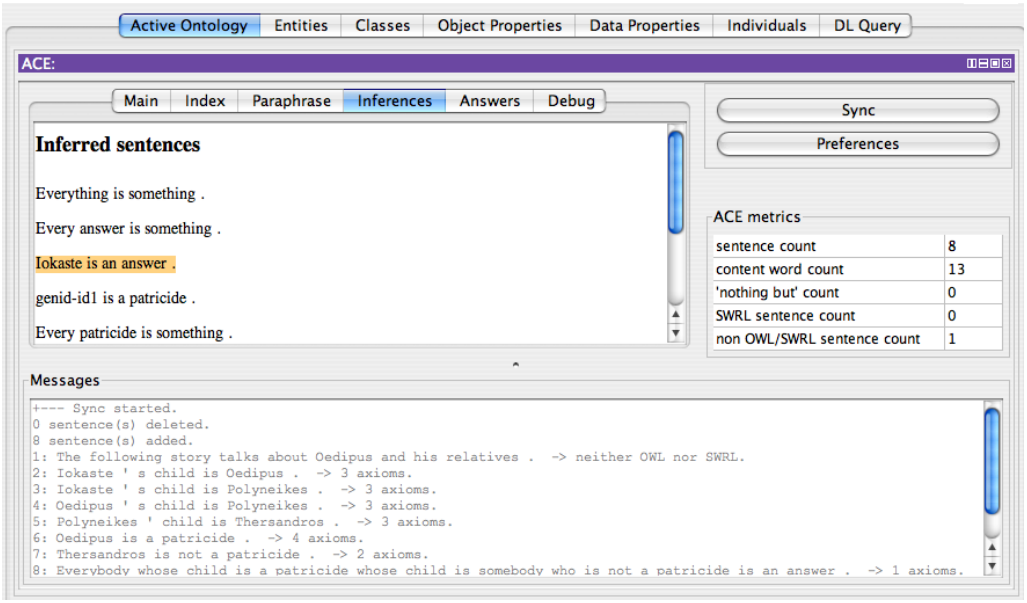


Figure 7.4: The “Inferences tab” shows a list of inferred axioms as ACE sentences. For example, the sentence “Iokaste is an answer.” was not explicitly stated in the original text (figure 7.2). Still, via non-trivial description logic reasoning this sentence can be derived.

Protégé also supports entailment explanations. Such an explanation is a sequence of axioms (usually previously asserted, but possibly synthesized) which motivates the entailment. The axioms in this sequence can be of any complexity and thus their natural language verbalization can bring significant improvement in understanding the reason behind the entailment. At the time of writing we have not implemented the explanation support yet.

7.4.7 Answers

The "Answers tab" is currently not implemented. It will allow an ACE question to be entered which would be translated into a DL Query and answered using the Protégé implementation of DL Query. The answers to a DL Query are named individuals (members of the queried class) or named classes (named super and sub classes of the queried class). In ACE terms, the answers are ACE content words — proper names and common nouns. While the answers to DL Queries are representation-wise identical in the ACE view and in the standard Protégé view, the construction of the query is potentially much simpler in the ACE view, as one has to construct a natural language question.

7.4.8 Debug

A "Debug tab" (figure 7.5) is currently provided to help the user get an overview of the logical and linguistic properties of the entered sentences. For sentences which have failed to map to OWL/SWRL, error messages are provided.

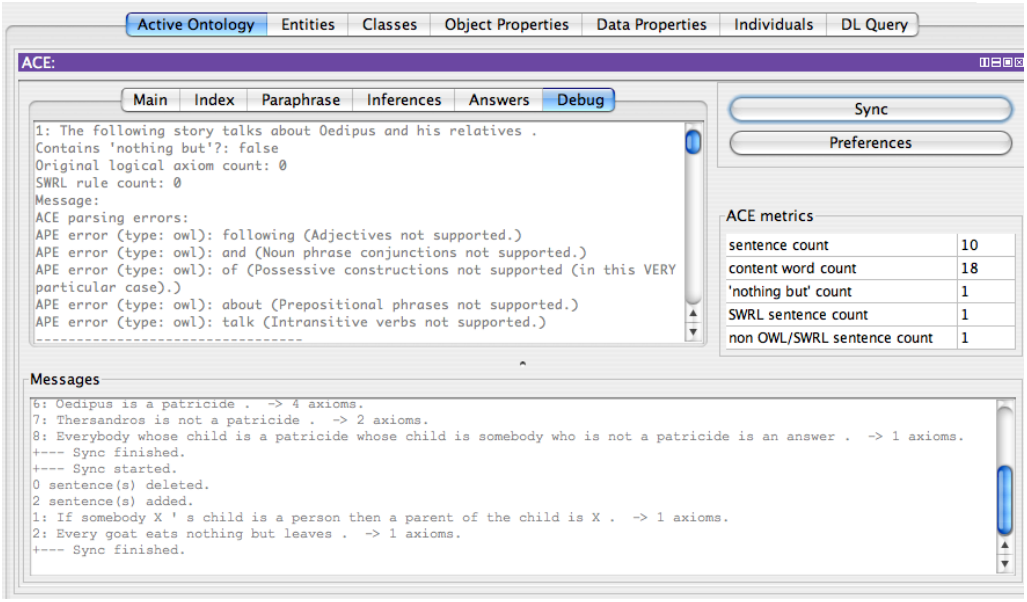


Figure 7.5: The "Debug tab" lists all the entered sentences along with their logical and linguistic properties. The sentence "The following story talks about Oedipus and his relatives." was not mapped to OWL/SWRL (i.e. it does not participate in the entailments) because the ACE→OWL currently does not support adjectives, prepositional phrases, noun phrase conjunction, and intransitive verbs.

Chapter 8

Evaluation

8.1 Introduction

There are several ways to evaluate the described work and the benefits that it might bring. The following lists some broad categories of possible evaluation methods.

- Evaluation of the described ACE fragments ACE_1 and ACE_2 in terms of the restrictions that they place on full ACE. Are these restrictions natural and easy to learn? Can these restrictions be easily explained in error messages, in case one violates the restrictions?
- Evaluation of ACE_1 and ACE_2 as alternative syntaxes for OWL. Are these alternative syntaxes "better" than traditional OWL syntaxes? One should study the readability, writability, learnability of the new syntaxes as compared to existing ones. The subjects of the usability tests would be both domain experts who need to use ontologies in their work, as well as OWL experts.
- Evaluation of ACE-based extensions to existing tools, e.g. the ACE View plug-in for Protégé (described in section 7.4). Do such extensions fit well into the existing tools? Do they make these tools easier to use? The subjects of such a study would be both professional knowledge engineers and domain experts, but also novice users of Protégé.
- Evaluation in the form of a case study of some real-world ontologies. In this evaluation, we can look at the verbalization of the ontologies and some of its characteristics, e.g. sentence lengths, number of similar sentence-patterns, the amount of axioms that are not supported by ACE_1 and ACE_2, etc. We can also study how many (previously undetected) modeling errors can domain experts discover and fix if they have the controlled natural language view to the ontology.

In this chapter, we perform the latter verbalization case study where we look at the natural language characteristics of the verbalization of existing real-world ontologies. As future work, we will extend this evaluation by the other listed methods, especially

Count	Axiom pattern
1223	DisjointClasses(_ _)
167	SubClassOf(_ _)
133	SubClassOf(_ ObjectSomeValuesFrom(_ _))
80	DataPropertyAssertion(_ _ _)
20	ObjectPropertyAssertion(_ _ _)
18	SubClassOf(_ ObjectSomeValuesFrom(_ ObjectUnionOf(_ _)))
13	SubClassOf(_ ObjectComplementOf(ObjectSomeValuesFrom(_ _)))
8	SubObjectPropertyOf(_ _)
7	FunctionalObjectProperty(_)
7	EquivalentClasses(_ ObjectIntersectionOf(_ ObjectSomeValuesFrom(_ _)))
6	SubClassOf(_ ObjectSomeValuesFrom(_ ObjectIntersectionOf(ObjectHasValue(_ _) _)))
6	SubClassOf(_ ObjectAllValuesFrom(_ ObjectUnionOf(_ _ _)))

Table 8.1: Frequency distribution of top 12 out of 81 axiom patterns in the Hydrology ontology. The underscores mark the presence of arbitrary named classes, properties, individuals, and data values.

when the ACE-based tools like ACE View have matured enough to provide a convenient evaluation environment where the subjects can be given a specific task and several alternative (i.e. competing) tools for solving the task.

8.2 Verbalization case study

8.2.1 Introduction

In this section, we study the verbalization of two real-world ontologies Hydrology and GALEN. Both of those ontologies feature reasonably complex axiom structures, i.e. we are not interested in ontologies which are simple directed acyclic graphs of *SubClassOf*-relations. We study some of the structural properties of the input axioms and the natural language characteristics of the resulting ACE sentences. Specifically, we are interested in how many real world axioms can be verbalized with our verbalization method and what are the reasons for a failure to verbalize. The implementation of the OWL verbalizer that was used in this experiment is briefly discussed in section 7.3.

8.2.2 Hydrology ontology

The Ordnance Survey Hydrology ontology[1] is a fairly complex ontology. It contains 1815 axioms. This number raises to 1858 after axiom-rewriting (described in 5.6.4) because some rewriting rules split axioms into several axioms. From the resulting axioms, 12 cannot be verbalized due to structural complexity (see 5.6.5). Most of the axioms are still quite simple and correspond to short ACE sentences. The ontology is verbalized in

less than 1 second on a modern laptop (Apple PowerBook G4).

The original axioms fall into 81 axiom patterns, most frequent of which is by far the *DisjointClasses*-axiom (table 8.1). The axioms exploit most of OWL's expressivity, e.g. usage of complementation and union is relatively frequent. The number of axioms containing the *ObjectAllValuesFrom*-description that is preserved after rewriting is 16, i.e. 16 sentences make use of the 'nothing but' marker discussed in section 5.7.6.

Figure 8.1 shows the frequency distribution of sentence lengths in the ACE verbalization of the Hydrology ontology.

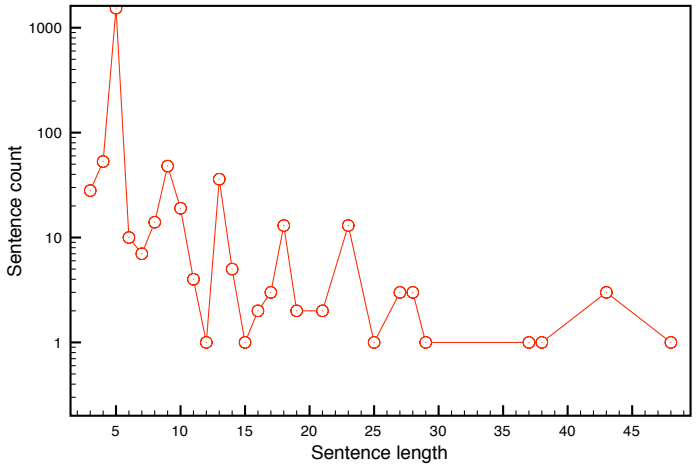


Figure 8.1: Frequency distribution of sentence lengths of the ACE verbalization of the Hydrology ontology. Most sentences are 5 tokens long and correspond (unsurprisingly) to the pattern "Every/No *ClassName1* is a *ClassName2*."

Even though most axioms are verbalized into structurally simple sentences containing just 5 tokens, there is a long tail of structurally more interesting and "longer" axioms. In the following examples, the definitions of classes *WaterWay*, *Aber*, and *DisusedCanal.Water* are shown.

$$\text{(8.1)} \quad \mathit{Waterway} \doteq (\mathit{River} \sqcup \mathit{Canal.Water}) \sqcap \exists\, \mathit{hasPrimaryPurpose}\, \mathit{InlandNavigation}$$

$$\text{(8.2)} \quad \mathit{Aber} \sqsubseteq \exists\, \mathit{hasRegionalSynonym}\, (\, (\mathit{RiverMouth} \sqcup \mathit{Confluence}) \sqcap \exists\, \mathit{hasDirectGeographicLocation}\, \{\mathit{Wales}\})$$

$$\text{(8.3)} \quad \mathit{DisusedCanal.Water} \doteq \mathit{BodyOfWater} \sqcap \neg(\exists\, \mathit{enables}\, \mathit{InlandNavigation}) \sqcap \exists\, \mathit{isDirectComponentOf}\, \mathit{DisusedCanalStretch} \sqcap$$

[1] `http://www.ordnancesurvey.co.uk/ontology/`

$$\exists\ containedIn\ Canal.Channel$$

The ACE counterparts of those axioms are given in the next examples. For the equivalence-axioms only one direction is given. Note the use of "comma and" to obtain the correct coordinator binding.

(8.4) Every Waterway is something that hasPrimaryPurpose an InlandNavigation, and that is a Canal.Water or that is a River.

(8.5) Every Aber hasRegionalSynonym something that hasDirectGeographicLocation Wales, and that is a Confluence or that is a RiverMouth.

(8.6) Every BodyOfWater that containedIn a Canal.Channel and that isDirectComponentOf a DisusedCanalStretch and that does not enables an InlandNavigation is a DisusedCanal.Water.

In this verbalization, OWL names are unedited and the morphological synthesis is defined by an "identity-lexicon" which maps word forms to themselves. This mapping also ignores the morphological properties of the OWL names, e.g. names that contain a period (e.g. 'Canal.Water') are actually not allowed in ACE. I.e. the author of the ontology must first make the names compatible with ACE and provide a lexicon that guides the morphological analysis/synthesis of the words, before he/she can convert the sentences back into OWL.

A few class descriptions in the Hydrology ontology are very complex and cannot be handled by our verbalization scheme. For example, the following class description embeds an intersection which embeds another intersection which embeds another intersection.

$$\begin{aligned}&\exists\ \text{hasRegion (SpatialRegion}\\ &\sqcap \neg (\exists\ \text{t\_spatiallyInside (SpatialRegion} \sqcap \exists\ \text{isRegionOf Sea)}\\ &\qquad \sqcap \exists\ \text{t\_spatiallyInside (SpatialRegion} \sqcap \exists\ \text{isRegionOf Lake))}\\ &\sqcap \exists\ \text{t\_spatiallyInside (SpatialRegion} \sqcap (\exists\ \text{isRegionOf (Sea} \sqcup \text{Lake))))}\end{aligned}$$

This structure is so complicated that it cannot be expressed with ACE relative clauses. It is nevertheless possible for the user to simplify this structure by using named classes (e.g. *SeaRegion*) that are defined to be equivalent to complex class descriptions, such as $(SpatialRegion \sqcap (\exists\ isRegionOf\ Sea))$.

8.2.3 GALEN ontology

GALEN is a medical ontology developed in the OpenGALEN project[2]. In the following, we use GALEN's translation into OWL which is provided by the CO-ODE project[3].

GALEN — one of the largest OWL ontologies currently available — contains 37,696 axioms. After axiom-rewriting the number of axioms raises to 48,139. Out of those,

[2] http://www.opengalen.org

[3] http://www.co-ode.org/galen/

Count	Axiom pattern
13338	SubClassOf(_ _)
10841	SubClassOf(_ ObjectSomeValuesFrom(_ _))
1855	EquivalentClasses(_ ObjectIntersectionOf(ObjectSomeValuesFrom(_ _) _))
1625	EquivalentClasses(ObjectIntersectionOf(ObjectSomeValuesFrom(_ _) _) _)
1566	EquivalentClasses(_ ObjectIntersectionOf(_ ObjectSomeValuesFrom(_ _))
1387	EquivalentClasses(ObjectIntersectionOf(_ ObjectSomeValuesFrom(_ _)) _)
958	SubObjectPropertyOf(_ _)
750	SubClassOf(_ ObjectSomeValuesFrom(_ ObjectIntersectionOf(ObjectSomeValuesFrom(_ _) _)))
475	InverseObjectProperties(_ _)
373	SubClassOf(_ ObjectSomeValuesFrom(_ ObjectIntersectionOf(_ ObjectSomeValuesFrom(_ _))))
337	InverseFunctionalObjectProperty(_)
337	FunctionalObjectProperty(_)

Table 8.2: Frequency distribution of top 12 out of 439 axiom patterns in the GALEN ontology. The underscores mark the presence of arbitrary named classes and object properties.

47,683 can be verbalized as ACE sentences and 456 are rejected as too complex (i.e. $\sim$1% of the axioms cannot be verbalized). It takes about 30 seconds on a modern laptop to produce the verbalization.

The original axioms fall into 430 different axiom patterns, the two most common patterns which greatly outnumber the rest are the *SubClassOf*-axiom between named classes, and an existential restriction between named classes (table 8.2). From the description logic point of view, GALEN is relatively simple — it does not contain *ObjectComplementOf*, *ObjectUnionOf*, *ObjectAllValuesFrom*, data properties, individuals, and cardinality restrictions beyond functionality of properties.

Figure 8.2 shows the frequency distribution of sentence lengths in the ACE verbalization of the GALEN ontology.

Because the axioms vary little and use only a small number of OWL constructs, the corresponding ACE sentences are rather repetitive. Although GALEN does not use explicit negation and disjunction, those constructs are conceptually still needed, as can be seen from class names like *nonNormal* and *MaleOrFemalePatient*. This also contributes to the poor readability of some of the verbalizations. The following are some sentence examples (starting from the longest sentence that contains 36 tokens).

(8.7) Every IschaemicCardiacPathology that isConsequenceOf a CoronaryArteryPathology and that hasSpecificSubprocess a StartingProcess that hasTimeOfOccurrence a TimeOfOccurrence that occursDuring an Age that hasQuantity a TemporalIntervalValue that hasMagnitude a lessThanForty and that hasUnit a years is an EarlyOnsetIschaemicCoronaryHeartDisease.

(8.8) Every VoluntaryEating that hasPathologicalStatus a pathological and that

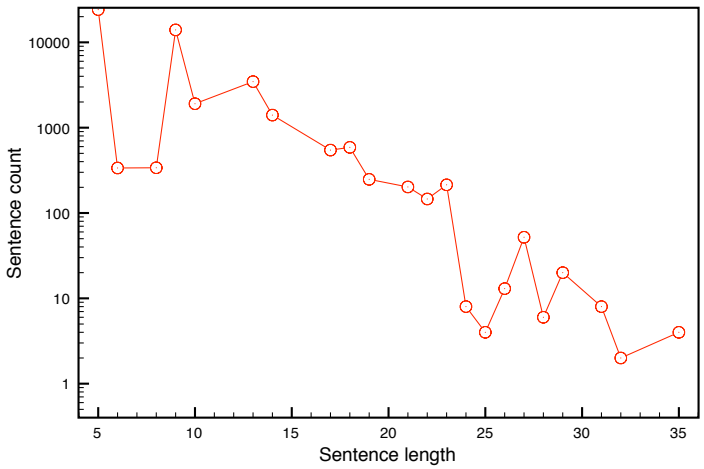


Figure 8.2: Frequency distribution of sentence lengths of the ACE verbalization of the GALEN ontology. Most sentences are 5 tokens long and correspond to the pattern "Every *ClassName1 Verb* a *ClassName2*.", where *Verb* is either the copula 'is' or a transitive verb corresponding to an object property. Another peak is at sentence length 9, where the sentence pattern is "Every *ClassName1* is a *ClassName2* that *Verb* a *ClassName2*."

actsSpecificallyOn a VitaminB1 that playsPhysiologicalRole a NutritiveFoodRole and that hasMass a Mass that hasQuantity a Level that hasMagnitude a lowLevel is a ThiamineDeficientNutrition.

(8.9) Every MaleOrFemalePatient is a Human that playsSocialRole a PatientRole and that hasPhenotypicalSex a PhenotypicalSex that hasAbsoluteState an unambiguousSex.

(8.10) Every ChemicalProcess that hasConsequence a pH that hasQuantity a Level that hasChangeInState a ChangeInLevelState is an AcidBaseReaction.

(8.11) Every NonNormallyLargeHand is a largeSize that isSizeOf a Hand and that hasAbnormalityStatus a nonNormal.

8.2.4 Conclusion

The results show that while most axioms in real-world ontologies are simple, such ontologies can contain a long tail of structurally very complex axioms. Only a few of such axioms are so complex that our proposed verbalization method cannot handle them, the rest, however, can be presented in a way that outperforms in readability the standard description logic syntax and its derivatives.

In our experiment, we verbalized the ontologies as they were, without performing any modification of the names of ontological entities. Changing the names to use conventional English capitalization and supplying the verbalizer with a lexicon of morphological mappings can further improve the verbalization. Note also that we presented each verbalization as a simple sequence of words. Those ACE-tools that have an option to use structure-aware layout in their presentation of ACE sentences can use line breaking and indentation to display better to scopes of coordinated relative clause chains, thereby making the sentences even more readable. Finally, we note that the verbalizer implementation is very fast as it can translate in a matter of seconds one of the largest OWL ontologies currently available.

Chapter 9

Conclusions and future work

In this thesis, we introduced the philosophy behind controlled natural languages and gave an overview of Attempto Controlled English (chapter 2). In chapter 3, we introduced the Semantic Web Ontology language OWL, concretely OWL 1.1 without data properties. We looked at various OWL syntaxes and OWL ontology editors, and discussed the difficulties that users face when working with OWL. In chapter 4, we reviewed the work related to using controlled English to read and write Semantic Web ontologies. In the main part of the thesis, in chapter 5, we described a novel solution to use Attempto Controlled English to express OWL in a bidirectional way, i.e. we used ACE both as an authoring and a verbalization language of ontologies that are semantically compatible with OWL. In chapter 6, we described various syntactically natural extensions of our approach to express OWL data properties, and rule and query languages. In chapter 7, we described an implementation of the ACE↔OWL mapping and how it can be used in a plug-in for the Protégé ontology editor. Finally, in chapter 8, we evaluated our solution for verbalizing OWL ontologies with a case study on two real-world ontologies.

We conclude that ACE can be used to write Semantic Web ontologies, meaning that ACE can express the content of OWL ontologies in a natural way, thus solving some usability issues with traditional OWL syntaxes. Furthermore, as an extension, we have shown that ACE scales naturally to express content usually found in business rules and queries. As a result, instead of using three different syntaxes, users can work via the unified "ACE interface".

We also conclude that existing OWL ontologies can be verbalized in concise and understandable English provided that a certain naming style is adopted for OWL individuals, classes, and properties, and that users develop lexicons to guide the morphological synthesis of the ACE surface forms. This result enables people with no background in formal logics to read and understand ontologies written by others, provided that the readers are experts in the domain described by the ontology. The major obstacle which has hindered this in the past, i.e. the formal and unknown syntax, has been removed in our approach by using a fragment of natural language as the syntax.

As future work, we intend to study and develop editors that support writing ACE ontologies, especially with the semantic aspects of this process. Such editors should have

a tight integration with ontology and rule reasoners, to be able to pinpoint reasons for inconsistencies or redundancies, and justify entailments in ACE texts. They should also have a seamless connection to query engines to be able to answer queries posed in ACE. While general OWL and SWRL reasoning tools already exist, the question remains how to best integrate them into the framework of natural language based ontology editing. We have already started work in this area by developing a simple plug-in for the Protégé ontology editor, which can be used to switch between the well-known Protégé-view and the new ACE-view to the ontology.

Another area of future research is to evaluate our approach more conclusively as outlined in chapter 8. This would enable us to pinpoint the exact target group that would profit from natural language based ontology engineering. A comparative evaluation of ACE is currently planned, where ACE is to be compared to other forms of controlled English, specifically Sydney OWL Syntax and Rabbit.

In this thesis, we used the latest version of ACE, i.e. version 6.0. Future versions of ACE might extend the language with syntactic sugar, i.e. constructs that are mapped to DRSs that are already handled by the mappings discussed in this thesis. Alternatively, the future versions might add constructs that increase the semantic expressivity of ACE and at the same time can be mapped to OWL, SWRL, etc., or to the extensions of those languages. In such cases, we intend to extend our mappings to handle such new ACE constructs.

Chapter 10

Summary in Estonian

Piiratud inglise keel ACE kui semantilise veebi keel. Kokkuvõte

Käesolevas väitekirjas vaadeldakse semantilise veebi (*Semantic Web*) keeli loomuliku keele vaatenurgast, täpsemalt piiratud inglise keele vaatenurgast. Väitekirjas kirjeldatakse automaatset ja süstemaatilist teisendust piiratud inglise keelest ACE (*Attempto Controlled English*) semantilise veebi ontoloogiakeelde OWL (*Web Ontology Language*) ning samuti pöördteisendust, keelest OWL keelde ACE. Tänu sellisele teisendusele muutub inimkauge süntaksiga keel OWL hõlpsasti kasutatavaks ka tavakasutajatele.

Semantilise veebi projektis on väljatöötatud hulgaliselt formaalseid loogikaid teadmiste automaatseks vahetamiseks ja omavaheliseks integreerimiseks. Projekti eesmärgiks on muuta veeb masinloetavaks, hõlbustades nõndaviisi inimeste suhtlust veebiga. Olgugi, et kavandatavas semantilises veebis on sama suur roll nii inimesel kui ka masinal, on väljatöötatud keeled — näiteks OWL ja SWRL (*Semantic Web Rule Language*) — küllalt inimvaenulikud, eeldades kasutajalt erialaseid teadmisi formaalloogikatest. Kuna antud keeled on oma olemuselt võrdlemisi väljendusrikkad (kattes suure fragmendi esimest järku predikaatloogikast), on neile kasutajaliideste loomine osutunud keerukaks ning pole andnud soovitud tulemusi — tavakasutajatel on ontoloogiate ja reeglistike lugemine ja kirjutamine ka spetsiaalsete liideste kaudu keeruline ning tihti loobutakse väljendusrikkamate konstruktsioonide kasutamisest.

Käesoleva töö panus on täpne kirjeldus teisendusest nende loogikate ja loomuliku keele vahel, kus loomuliku keele osas on inimloetav kuid samas täpne ja mitmesusevaba piiratud inglise keel. Kuna loomulikul keelel on näiteks graafiliste kasutajaliideste ees mitmeid eeliseid, loovad antud töö tulemused võimaluse uuteks ja palju kasutajasõbralikumateks ontoloogia- ja reegliredaktoriteks.

Peatükis 2 tutvustame keelt ACE, selle konstruktsiooni- ja interpretatsioonireegleid ning teisendust predikaatloogikasse. Peatükis 3 kirjeldame semantilise veebi ontoloogiakeelt OWL, selle keele erinevaid esitusvorme ja redaktoreid, ning viitame keele puudustele tavakasutaja seisukohalt. Peatükis 4 vaatleme olemasolevate uuringute tulemusi, mis on meie väitekirjaga seotud. Töö põhiosas, peatükis 5, kirjeldame formaalset teisendust keelest ACE keelde OWL ning vastupidi, keelest OWL keelde ACE. Peatükis 6 laiendame oma lähenemist reeglite keelele SWRL ja teadmusbaasi päringuformalis-

midele, mis on keelest OWL süntaktiliselt väga erinevad. Samas jääb keele ACE süntaks peaaegu muutumatuks. Peatükis 7 tutvustame käsitletud teisenduste implementatsiooni ning nende kasutust ontoloogiaredaktoris Protégé. Lõpuks, peatükis 8, vaatleme kaht reaalses kasutuses olevat ontoloogiat loodud teisenduste vaatenurgast.

Selle töö järelduseks on, et semantilise veebi jaoks vajalikke masinmõistetavaid ontoloogiaid ja reeglisüsteeme ning päringuid nendele saab edukalt kirjeldada piiratud loomulikus keeles, muutes formaalsed semantilise veebi keeled ja neis kirjutatu nõnda kättesaadavaks ka tavakasutajatele.

Chapter 11

Acknowledgments

First of all, I would like to thank Norbert E. Fuchs who got me started in the field of controlled natural languages, who agreed to become my supervisor, and who provided constant support throughout my doctoral studies.

I would also like to thank Kaili Müürisep for taking over the co-supervisor task.

I would like to thank professor Mare Koit, for her constant support throughout my career as a computational linguist.

I am grateful to professor Michael Hess and the Institute of Computational Linguistics led by him at the Department of Informatics, University of Zurich, for providing a stimulating working environment.

I would like to thank all the people together with whom I have worked in the Attempto project over the years, especially Tobias Kuhn, Gerold Schneider, and Fabio Rinaldi. They made many useful comments on the various drafts of this thesis.

I would also like to thank the developers of other controlled forms of English, namely Rolf Schwitter, Anne Cregan (the developers of *Sydney OWL Syntax*), and Cathy Dolbear, Glen Hart (the developers of *Rabbit*) for fruitful discussions about a possible controlled English based front-end to OWL.

Finally, I am grateful to my parents, my brother and my girlfriend, and to all my friends in Tallinn, Zürich, and Tartu.

This research has been funded by the European Commission and by the Swiss State Secretariat for Education and Research within the 6th Framework Program project REWERSE number 506779 (cf. `http://rewerse.net`).

Bibliography

[BB99] Patrick Blackburn and Johan Bos. *Working with Discourse Representation Structures*, volume 2nd of *Representation and Inference for Natural Language: A First Course in Computational Linguistics*. September 1999.

[BCM+03] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook. Theory, Implementation and Applications*. Cambridge University Press, 2003.

[BCT07] Raffaella Bernardi, Diego Calvanese, and Camilo Thorne. Lite Natural Language. In *IWCS-7*, 2007.

[BKFvB04] Abraham Bernstein, Esther Kaufmann, Norbert E. Fuchs, and June von Bonin. Talking to the Semantic Web — A Controlled English Query Interface for Ontologies. In *14th Workshop on Information Technology and Systems*, pages 212–217, December 2004.

[BM04] Paul V. Biron and Ashok Malhotra. XML Schema Part 2: Datatypes Second Edition. W3C Recommendation 28 October 2004. Technical report, W3C, 2004.

[BMPS+91] Ronald J. Brachman, Deborah L. McGuinness, Peter F. Patel-Schneider, Lori Alperin Resnick, and Alexander Borgida. Living with CLASSIC: When and How to Use a KL-ONE-Like Language. *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 401–456, 1991.

[CD99] James Clark and Steve DeRose. XML Path Language (XPath). Version 1.0. W3C Recommendation 16 November 1999. Technical report, W3C, 1999.

[CGL+05] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. DL-Lite: Tractable description logics for ontologies. In *AAAI 2005*, pages 602–607, 2005.

[CHJ+05] Peter Clark, Philip Harrison, Thomas Jenkins, John Thompson, and Richard H. Wojcik. Acquiring and Using World Knowledge Using a Restricted Subset of English. In *FLAIRS 2005*, pages 506–511, 2005.

[Cla99] James Clark. XSL Transformations (XSLT). Version 1.0. W3C Recommendation 16 November 1999. Technical report, W3C, 1999.

[CP04] Peter Clark and Bruce Porter. KM — The Knowledge Machine 2.0: Users Manual. Technical report, 2004. `http://www.cs.utexas.edu/users/mfkb/km/userman.pdf`.

[CSM07] Anne Cregan, Rolf Schwitter, and Thomas Meyer. Sydney OWL Syntax — towards a Controlled Natural Language Syntax for OWL 1.1. In Christine Golbreich, Aditya Kalyanpur, and Bijan Parsia, editors, *3rd OWL Experiences and Directions Workshop (OWLED 2007)*, volume 258. CEUR Proceedings, 2007.

[DHK+07] Catherine Dolbear, Glen Hart, Katalin Kovacs, Sheng Zhou, and John Goodwin. The Rabbit Language: Description, Syntax and Conversion to OWL. Technical Report I-0004, Ordnance Survey Research, 2007. `http://www.ordnancesurvey.co.uk/oswebsite/partnerships/research/publications/docs/2007/Rabbit_Language_v1.pdf`.

[DMB+06] Martin Dzbor, Enrico Motta, Carlos Buil, Jose Gomez, Olaf Görlitz, and Holger Lewen. Developing ontologies in OWL: An observational study. In *2nd OWL Experiences and Directions Workshop (OWLED 2006)*, 2006.

[FDT+07] Adam Funk, Brian Davis, Valentin Tablan, Kalina Bontcheva, and Hamish Cunningham. D2.2.2 Report: Controlled Language IE Components version 2. Technical report, University of Sheffield, 2007.

[FHH03] Richard Fikes, Pat Hayes, and Ian Horrocks. OWL-QL: A Language for Deductive Query Answering on the Semantic Web. KSL Technical Report 03-14. Technical report, Stanford Knowledge Systems Laboratory, Stanford University, Stanford, CA, USA, 2003.

[FKK06a] Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Deliverable I2-D9. Attempto Controlled English 5: Language Extensions and Tools I. Technical report, REWERSE, 2006. 28 pages, `http://rewerse.net/deliverables.html`.

[FKK+06b] Norbert E. Fuchs, Kaarel Kaljurand, Tobias Kuhn, Gerold Schneider, Loic Royer, and Michael Schröder. Deliverable I2-D7. Attempto Controlled English and the Semantic Web. Technical report, REWERSE, 2006. 28 pages, `http://rewerse.net/deliverables.html`.

[FKK07a] Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. ACE 6.0 Construction Rules. Technical report, Attempto project, 2007. http://attempto.ifi.uzh.ch/site/docs/ace/6.0/ace_constructionrules.html.

[FKK07b] Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. ACE 6.0 Interpretation Rules. Technical report, Attempto project, 2007. http://attempto.ifi.uzh.ch/site/docs/ace/6.0/ace_interpretationrules.html.

[FKK07c] Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. ACE 6.0 Syntax Report. Technical report, Attempto project, 2007. http://attempto.ifi.uzh.ch/site/docs/ace/6.0/syntax_report.html.

[FKK07d] Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Deliverable I2-D11. Attempto Controlled English 5: Language Extensions and Tools II. Technical report, REWERSE, 2007. 33 pages, http://rewerse.net/deliverables.html.

[FKK07e] Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Deliverable I2-D13. Reasoning, Rules and Semantic Wikis. Technical report, REWERSE, 2007. 26 pages, http://rewerse.net/deliverables.html.

[FKK07f] Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Discourse Representation Structures for ACE 5.5. Technical Report ifi-2007.05, Department of Informatics, University of Zurich, Zurich, Switzerland, 2007. 55 pages.

[FKS05] Norbert E. Fuchs, Kaarel Kaljurand, and Gerold Schneider. Deliverable I2-D5. Verbalising Formal Languages in Attempto Controlled English I. Technical report, REWERSE, 2005. 21 pages, http://rewerse.net/deliverables.html.

[FKS06] Norbert E. Fuchs, Kaarel Kaljurand, and Gerold Schneider. Attempto Controlled English Meets the Challenges of Knowledge Representation, Reasoning, Interoperability and User Interfaces. In Geoff Sutcliffe and Randy Goebel, editors, *Nineteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS 2006)*, pages 664–669, Melbourne Beach, Florida, May 11–13th 2006. The AAAI Press, Menlo Park, California.

[FS03] Norbert E. Fuchs and Uta Schwertel. Reasoning in Attempto Controlled English. In Francois Bry, Nicola Henze, and Jan Małuszyński, editors, *Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR 2003)*, number 2901 in Lecture Notes in Computer Science. Springer, 2003.

[GHP+06] Bernardo Cuenca Grau, Ian Horrocks, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler. Next Steps for OWL. In *2nd OWL Experiences and Directions Workshop (OWLED 2006)*, 2006.

[GHVD03] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: Combining logic programs with description logic. In *Twelfth International World Wide Web Conference (WWW 2003)*, pages 48–57, Budapest, Hungary, May 20–24th 2003.

[Gib98] Edward Gibson. Linguistic complexity: locality of syntactic dependencies. *Cognition*, 68(1):1–76, 1998.

[Gli04] Birte Glimm. A query language for web ontologies. Technical report, Hamburg University of Applied Sciences, 2004. Bachelor Report.

[GM07] Bernardo Cuenca Grau and Boris Motik. OWL 1.1 Web Ontology Language Model-Theoretic Semantics. Technical report, 2007. `http://webont.org/owl/1.1/semantics.html`.

[GMPS07] Bernardo Cuenca Grau, Boris Motik, and Peter Patel-Schneider. OWL 1.1 Web Ontology Language XML Syntax. Technical report, 2007. `http://webont.org/owl/1.1/xml_syntax.html`.

[Goo07] John Goodwin. A Methodology for Converting Conceptual Ontologies to OWL. Technical Report I-0005, Ordnance Survey Research, 2007. `http://www.ordnancesurvey.co.uk/oswebsite/partnerships/research/publications/docs/2007/Methodology_for_converting_conceptual_ontologies_to_OWL_V1.pdf`.

[Gra07] Bernardo Cuenca Grau. OWL 1.1 Web Ontology Language Tractable Fragments. Technical report, 2007. `http://webont.org/owl/1.1/tractable.html`.

[Hal01] Terry A. Halpin. *Information modeling and relational databases*. Morgan Kaufmann Publishers, San Francisco, 3rd edition, 2001.

[HBN07] Matthew Horridge, Sean Bechhofer, and Olaf Noppens. Igniting the OWL 1.1 Touch Paper: The OWL API. In Christine Golbreich, Aditya Kalyanpur, and Bijan Parsia, editors, *3rd OWL Experiences and Directions Workshop (OWLED 2007)*, volume 258. CEUR Proceedings, 2007.

[HDG+06] Matthew Horridge, Nick Drummond, John Goodwin, Alan Rector, Robert Stevens, and Hai H. Wang. The Manchester OWL Syntax. In *2nd OWL Experiences and Directions Workshop (OWLED 2006)*, 2006.

[HDG07] Glen Hart, Catherine Dolbear, and John Goodwin. Lege Feliciter: Using Structured English to represent a Topographic Hydrology Ontology. In Christine Golbreich, Aditya Kalyanpur, and Bijan Parsia, editors, *3rd*

OWL Experiences and Directions Workshop (OWLED 2007), volume 258. CEUR Proceedings, 2007.

[HDGK07] Glen Hart, Catherine Dolbear, John Goodwin, and Katalin Kovacs. Domain Ontology Development. Technical Report I-0003, Ordnance Survey Research, 2007. http://www.ordnancesurvey.co.uk/oswebsite/partnerships/research/publications/docs/2007/Domain_Ontology_Development_V1.pdf.

[Hef04] Jeff Heflin. OWL Web Ontology Language Use Cases and Requirements. W3C Recommendation 10 February 2004. Technical report, W3C, 2004.

[HG07] Glen Hart and John Goodwin. Modelling Guidelines for Constructing Domain Ontologies. Technical Report I-0006, Ordnance Survey Research, 2007. http://www.ordnancesurvey.co.uk/oswebsite/partnerships/research/publications/docs/2007/Modelling_Guidelines_V1.pdf.

[HJM+07] Matthew Horridge, Simon Jupp, Georgina Moulton, Alan Rector, Robert Stevens, and Chris Wroe. A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools. Edition 1.1. Technical report, The University Of Manchester, 2007. http://www.co-ode.org/resources/tutorials/.

[HKKHW05] Daniel Hewlett, Aditya Kalyanpur, Vladimir Kolovski, and Chris Halaschek-Wiener. Effective Natural Language Paraphrasing of Ontologies on the Semantic Web. In *End User Semantic Web Interaction Workshop (ISWC 2005)*, 2005.

[HKS06] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible $\mathcal{SROIQ}$. In *KR 2006*, 2006.

[HMW04] Volker Haarslev, Ralf Möller, and Michael Wessel. Querying the Semantic Web with Racer + nRQL. *KI-2004 International Workshop on Applications of Description Logics (ADL 2004)*, 2004.

[Höf04] Stefan Höfler. The Syntax of Attempto Controlled English: An Abstract Grammar for ACE 4.0. Technical Report ifi-2004.03, Department of Informatics, University of Zurich, Zurich, Switzerland, 2004.

[Hor03] Ian Horrocks. Implementation and Optimisation Techniques. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors, *The Description Logic Handbook. Theory, Implementation and Applications*, pages 313–355. Cambridge University Press, 2003.

[HPSB+04] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission 21 May 2004. Technical report, W3C, 2004. http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.

[HT00] Ian Horrocks and Sergio Tessaris. A Conjunctive Query Language for Description Logic Aboxes. In *Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 399–404. AAAI Press / The MIT Press, 2000.

[HT02] Ian Horrocks and Sergio Tessaris. Querying the Semantic Web: a Formal Approach. In Ian Horrocks and James Hendler, editors, *ISWC 2002*, number 2342 in Lecture Notes in Computer Science, pages 177–191. Springer, 2002.

[HWGP+06] Christian Halaschek-Wiener, Jennifer Golbeck, Bijan Parsia, Vladimir Kolovski, and Jim Hendler. Image browsing and natural language paraphrases of semantic web annotations. In *First International Workshop on Semantic Web Annotations for Multimedia (SWAMM)*, Edinburgh, Scotland, May 22nd 2006.

[JKD06] Mustafa Jarrar, Maria Keet, and Paolo Dongilli. Multilingual verbalization of ORM conceptual models and axiomatized ontologies. Technical report, Vrije Universiteit Brussel, February 2006.

[Kam81] Hans Kamp. A Theory of Truth and Semantic Representation. In J. Groenendijk, T. Janssen, and M. Stokhof, editors, *Formal Methods in the Study of Language*, pages 277–322. Mathematical Centre, Amsterdam, 1981.

[KF06a] Kaarel Kaljurand and Norbert E. Fuchs. Bidirectional mapping between OWL DL and Attempto Controlled English. In José Júlio Alferes, James Bailey, Wolfgang May, and Uta Schwertel, editors, *Fourth Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR 2006)*, number 4187 in Lecture Notes in Computer Science, pages 179–189, Budva, Montenegro, 2006. Springer.

[KF06b] Kaarel Kaljurand and Norbert E. Fuchs. Mapping Attempto Controlled English to OWL DL. In *3rd European Semantic Web Conference. Demo and Poster Session*, pages 11–12, Budva, Montenegro, June 12th 2006.

[KF07] Kaarel Kaljurand and Norbert Fuchs. Verbalizing OWL in Attempto Controlled English. In Christine Golbreich, Aditya Kalyanpur, and Bijan Parsia, editors, *3rd OWL Experiences and Directions Workshop (OWLED 2007)*, volume 258. CEUR Proceedings, 2007. 10 pages.

[KPSG06] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and Bernardo Cuenca Grau. Repairing Unsatisfiable Concepts in OWL Ontologies. In *ESWC 2006*, 2006.

[KR93] Hans Kamp and Uwe Reyle. *From Discourse to Logic. Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Kluwer Academic Publishers, Dordrecht/Boston/London, 1993.

[KSSP07] Alexander Kubias, Simon Schenk, Steffen Staab, and Jeff Z. Pan. OWL SAIQL — An OWL DL Query Language for Ontology Extraction. In Christine Golbreich, Aditya Kalyanpur, and Bijan Parsia, editors, *3rd OWL Experiences and Directions Workshop (OWLED 2007)*, volume 258. CEUR Proceedings, 2007.

[Kuh07] Tobias Kuhn. AceRules: Executing Rules in Controlled Natural Language. In Massimo Marchiori, Jeff Z. Pan, and Christian de Sainte Marie, editors, *First International Conference on Web Reasoning and Rule Systems (RR2007)*, Lecture Notes in Computer Science, pages 299–308. Springer, 2007.

[Lew96] Richard L. Lewis. Interference in short-term memory: The magical number two (or three) in sentence processing. *Journal of Psycholinguistic Research*, 25(1):93–115, January 1996.

[LW06] Sergey Lukichev and Gerd Wagner. Deliverable I1-D6. Verbalization of the REWERSE I1 Rule Markup Language. Technical report, REWERSE, 2006. `http://rewerse.net/deliverables.html`.

[MMW07] Ashok Malhotra, Jim Melton, and Norman Walsh. XQuery 1.0 and XPath 2.0 Functions and Operators. W3C Recommendation 23 January 2007. Technical report, W3C, 2007. `http://www.w3.org/TR/xpath-functions/`.

[MP07] Chris Mellish and Jeff Z. Pan. Natural Language Directed Inference from Ontologies. 2007.

[MPSH07] Boris Motik, Peter F. Patel-Schneider, and Ian Horrocks. OWL 1.1 Web Ontology Language Structural Specification and Functional-Style Syntax. Technical report, 2007. `http://www.webont.org/owl/1.1/owl_specification.html`.

[MS05a] Chris Mellish and Xiantang Sun. Natural Language Directed Inference in the Presentation of Ontologies. In *10th European Workshop on Natural Language Generation*, Aberdeen, Scotland, August 8–10th 2005.

[MS05b] Chris Mellish and Xiantang Sun. The Semantic Web as a Linguistic Resource. In *Twenty-sixth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, Peterhouse College, Cambridge, UK, December 12–14th 2005.

[NM01] Natalya F. Noy and Deborah L. McGuinness. Ontology Development 101: A Guide to Creating Your First Ontology. Technical report, Stanford Knowledge Systems Laboratory, 2001.

[PH03] Ian Pratt-Hartmann. A two-variable fragment of English. *Journal of Logic, Language and Information*, 12(1):13–45, 2003.

[PH04] Ian Pratt-Hartmann. Fragments of Language. *Journal of Logic, Language and Information*, 13(2):207–223, 2004.

[PHT06] Ian Pratt-Hartmann and Allan Third. More fragments of language: the case of ditransitive verbs. *Notre Dame Journal of Formal Logic*, 47(2):151–177, 2006.

[Poo06] Jonathan Pool. Can Controlled Languages Scale to the Web? In *5th International Workshop on Controlled Language Applications*, 2006.

[PS07] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Candidate Recommendation 14 June 2007. Technical report, W3C, 2007.

[PSG+05] Bijan Parsia, Evren Sirin, Bernardo Cuenca Grau, Edna Ruckhaus, and Daniel Hewlett. Cautiously Approaching SWRL. Technical report, University of Maryland, 2005.

[RDH+04] Alan L. Rector, Nick Drummond, Matthew Horridge, Jeremy Rogers, Holger Knublauch, Robert Stevens, Hai Wang, and Chris Wroe. OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns. In Enrico Motta, Nigel Shadbolt, Arthur Stutt, and Nicholas Gibbins, editors, *Engineering Knowledge in the Age of the Semantic Web, 14th International Conference, EKAW 2004*, volume 3257 of *Lecture Notes in Computer Science*, pages 63–81, Whittlebury Hall, UK, October 5–8th 2004. Springer.

[Sch04] Uta Schwertel. *Plural Semantics for Natural Language Understanding — A Computational Proof-Theoretic Approach*. PhD thesis, University of Zurich, 2004.

[Sch05a] Rolf Schwitter. A Controlled Natural Language Layer for the Semantic Web. In S. Zhang and R. Jarvis, editors, *AI 2005, Advances in Artificial Intelligence: 18th Australian Joint Conference on Artificial Intelligence*, pages 425–434, December 2005.

[Sch05b] Rolf Schwitter. Controlled Natural Language as Interface Language to the Semantic Web. In *2nd Indian International Conference on Artificial Intelligence (IICAI-05)*, Pune, India, December 20–22nd 2005.

[She07] Rob Shearer. Structured Ontology Format. In Christine Golbreich, Aditya Kalyanpur, and Bijan Parsia, editors, *3rd OWL Experiences and Directions Workshop (OWLED 2007)*, volume 258. CEUR Proceedings, 2007.

[SLH03] Rolf Schwitter, Anna Ljungberg, and David Hood. ECOLE — A Look-ahead Editor for a Controlled Language. In *Controlled Translation, Proceedings of EAMT-CLAW03, Joint Conference combining the 8th International Workshop of the European Association for Machine Translation and the 4th Controlled Language Application Workshop*, pages 141–150, Dublin City University, Ireland, May 15–17th 2003.

[Sow04] John F. Sowa. Common Logic Controlled English. Technical report, 2004. Draft, 24 February 2004, `http://www.jfsowa.com/clce/specs.htm`.

[Sow05] John F. Sowa. Propositions. Technical report, May 2005. `http://www.jfsowa.com/logic/proposit.htm`.

[Sow07] John F. Sowa. Common Logic Controlled English. Technical report, 2007. Draft, 15 March 2007, `http://www.jfsowa.com/clce/clce07.htm`.

[SP07] Evren Sirin and Bijan Parsia. SPARQL-DL: SPARQL Query for OWL-DL. In Christine Golbreich, Aditya Kalyanpur, and Bijan Parsia, editors, *3rd OWL Experiences and Directions Workshop (OWLED 2007)*, volume 258. CEUR Proceedings, 2007.

[SPG+07] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A Practical OWL-DL Reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, 2007.

[ST04] Rolf Schwitter and Marc Tilbrook. Controlled Natural Language meets the Semantic Web. In S. Wan A. Asudeh, C. Paris, editor, *Australasian Language Technology Workshop 2004*, pages 55–62, Macquarie University, December 2004.

[ST06] Rolf Schwitter and Marc Tilbrook. Let's Talk in Description Logic via Controlled Natural Language. In *Logic and Engineering of Natural Language Semantics 2006, (LENLS2006)*, Tokyo, Japan, June 5–6th 2006.

[Tes01] Sergio Tessaris. *Questions and answers: reasoning and querying in Description Logic*. PhD thesis, University of Manchester, 2001.

[TH06] Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *International Joint Conference on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.

[Thi06] Allan Third. *Logical analysis of fragments of natural language*. PhD thesis, University of Manchester, 2006.

[Vra05] Denny Vrandecic. Explicit knowledge engineering patterns with macros. In Chris Welty and Aldo Gangemi, editors, *Ontology Patterns for the Semantic Web Workshop at the ISWC 2005*, 2005.

[WCH87] Morton E. Winston, Roger Chaffin, and Douglas Herrmann. A taxonomy of Part-Whole relations. *Cognitive Science*, 11:417–444, 1987.

[Wil03] Graham Wilcock. Talking OWLs: Towards an Ontology Verbalizer. In *ISWC 2003. Human Language Technology for the Semantic Web and Web Services*, pages 109–112, Sanibel Island, Florida, 2003.

Curriculum vitae

Curriculum vitae in English

General information

1. Name: Kaarel Kaljurand
2. Date and place of birth: 28.10.1975, Tallinn
3. Citizenship: Estonia
4. Marital status: single
5. Contact data:
   - Sõpruse puiestee 246-67, Tallinn, Estonia
   - +372 56 50 29 38
   - kaljurand@gmail.com
6. Current position: research assistant at the University of Zurich, Department of Informatics
7. Education
   - 1994. finished Tallinn 44^{th} Upper Secondary School (currently called Mustamäe Gümnaasium)
   - 1994–1995 Foothill College, Los Altos, California, USA
   - 2000. BSc in Computer Science, University of Tartu, Department of Mathematics, Institute of Computer Science. Academic degree: *baccalaureus scientiarum* in Computer Science, University of Tartu, 21. June 2000.
   - 2002. MSc in Computer Science, University of Tartu, Department of Mathematics and Computer Science, Institute of Computer Science. Academic degree: *magister scientiarum* in Computer Science, University of Tartu, 4. June 2002. Topic: "Automatic terminology extraction" (in Estonian)
   - 2002– PhD studies, University of Tartu, Department of Mathematics and Computer Science, Institute of Computer Science

- Oct 2002–Dec 2003 Guest at the University of Zurich, Institute of Computational Linguistics

8. Language skills: Estonian (native language), English (fluent), German (reading skills)

9. Professional employment:

   - Nov 1999 – Oct 2001 — Programmer and researcher in "Answer extraction from electronic documents in linguistics based information retrieval", joint-project between University of Zurich and University of Tartu
   - 2001–2004 — Involvement in several projects dealing with extending the Estonian corpora and developing a Word Sense Disambiguation system for Estonian
   - July 2004 – June 2005 — Research assistant at the University of Zurich, Department of Informatics. Involvement in project Parmenides. Ontology driven Temporal Text mining on organizational data for extracting temporal valid knowledge
   - Since October 2004 — Research assistant at the University of Zurich, Department of Informatics. Involvement in project REWERSE (Reasoning on the Web with Rules and Semantics), `http://rewerse.net`

Scientific activities

1. Research interests

   - Controlled Natural Languages
   - Semantic Web, OWL ontology language
   - Corpus linguistics (Treebanks)
   - Word Sense Disambiguation (of Estonian)
   - Syntactic analysis of Estonian (Dependency Syntax)

2. Grants

   - Swiss scholarship for university studies, fine arts and music schools (*Stipendium für ausländische Studierende und Kunstschaffende in der Schweiz*) to study at the University of Zurich during the school-year 2002/2003 and winter semester 2003 (11.5 months in total)
   - EITF (Estonian Information Technology Foundation) Tiger University grant to support my PhD studies at University of Tartu during the school-year 2003/2004
   - ESSLLI 2004 grant, sponsored by the ESSLLI 2004 local organizers, to cover the registration fee, accomodation and lunch at ESSLLI 2004

- OWLED 2007 grant, funded by the European Commission 6th Framework Programme project Knowledge Web, to cover the registration fee, accomodation, and travel costs of the participation at OWLED 2007

Additional studies

- 23.08–05.09.1998 summer school "Formal Grammars and their Applications", University of Tartu, Tartu
- 01.–12.03.1999 14th Vilem Mathesius Lecture Series, Charles University, Prague
- 04.–09.03.2001 6th Estonian Winter School in Computer Science, Palmse
- 23.07–03.08.2001 8th Central European Summer School in Generative Grammar, Nish
- 13–24.08.2001 13th European Summer School in Logic, Language and Information, Helsinki
- 18–29.08.2003 15th European Summer School in Logic, Language and Information, Vienna
- 01–05.09.2003 Course "Language evolution and computation", Simon Kirby and Jim Hurford (University of Edinburgh). Graduate School of Language Technology, Helsinki
- 30.01–01.02.2004 Estonian Computer Science Theory Days, Koke
- 01–05.03.2004 PhD course "Treebanks: Formats, Tools and Usage", Stockholm University, Stockholm
- 07–20.03.2004 19th Vilem Mathesius Lecture Series, Charles University, Prague
- 09–20.08.2004 16th European Summer School in Logic, Language and Information, Nancy
- 25–29.07.2005 Reasoning Web 2005 Summer School, University of Malta, Msida

Curriculum vitae in Estonian

Üldandmed

1. Ees- ja perekonnanimi: Kaarel Kaljurand
2. Sünniaeg ja koht: 28.10.1975, Tallinn
3. Kodakondsus: Eesti Vabariik
4. Perekonnaseis: vallaline

5. Aadress, telefon, *e-mail*

    - Sõpruse puiestee 246-67, Tallinn, Eesti Vabariik
    - +372 56 50 29 38
    - kaljurand@gmail.com

6. Praegune töökoht, amet

    - Zürichi Ülikool, teadur

7. Haridus

    - 1994. lõpetatud Tallinna 44. Keskkool (praegune Mustamäe Gümnaasium)
    - 1994–1995 Foothill College, Los Altos, California, USA
    - 2000. BSc informaatikas, Tartu Ülikool, matemaatika-informaatikateaduskond. Kraad: *baccalaureus scientiarum* informaatika erialal, Tartu Ülikool, 21. juunil 2000. aastal.
    - 2002. MSc informaatikas, Tartu Ülikool, matemaatika-informaatikateaduskond. Kraad: *magister scientiarum* informaatika erialal, Tartu Ülikool, 4. junil 2002. aastal. Teema: "Automaatne terminite tuvastamine"
    - 2002– doktoriõpingud, Tartu Ülikool, matemaatika-informaatikateaduskond
    - okt 2002–dets 2003 külalisdoktorandina Zürichi Ülikoolis, Arvutilingvistika instituudis

8. Keelteoskus: eesti (emakeel), inglise (kõrgtase), saksa (kesktase)

9. Töökogemus:

    - november 1999 – oktoober 2001 — programmeerija, Tartu Ülikool, Arvutiteaduse instituut. Osalemine Zürichi Ülikooli ja Tartu Ülikooli ühisprojektis "Answer extraction from electronic documents in linguistics based information retrieval"
    - 2001–2004 — osalemine erinevates projektides eesti keele korpuse täiendamiseks ning eesti keele sõnatähenduste ühestaja loomiseks
    - juuli 2004 – juuni 2005 — teadur, Zürichi Ülikool, Arvutilingvistika instituut. Osalemine projektis "Parmenides" (*Ontology driven Temporal Text mining on organizational data for extracting temporal valid knowledge*)
    - oktoober 2004 — teadur, Zürichi Ülikool, Arvutilingvistika instituut. Osalemine projektis "REWERSE" (*Reasoning on the Web with Rules and Semantics*)

Teaduslik ja arendustegevus

1. Peamised uurimisvaldkonnad

    - piiratud loomulikud keeled
    - Semantiline veeb, ontoloogia keel OWL (*Web Ontology Language*)
    - korpuslingvistika (puudepangad)
    - eesti keele sõnatähenduste automaatne ühestamine
    - eesti keele süntaktiline analüüs (sõltuvussüntaks)

2. Saadud uurimistoetused ja stipendiumid

    - Sveitsi valitsuse stipendium (*Stipendium für ausländische Studierende und Kunstschaffende in der Schweiz*) toetamaks minu õpinguid Zürichi Ülikoolis õppeaastal 2002/2003 ja õppeaasta 2003/2004 talvesemestril (kokku 11,5 kuud)
    - Eesti Infotehnoloogia Sihtasutuse (EITSA) Tiigriülikooli stipendium toetamaks minu doktoriõpinguid Tartu Ülikoolis õppeaastal 2003/2004
    - ESSLLI 2004 stipendium, finantseeritud ESSLLI 2004 organiseerijate poolt, katmaks minu osavõtukulusid suvekoolis ESSLLI 2004
    - OWLED 2007 stipendium, finantseeritud projekti "Knowledge Web" poolt, katmaks minu osavõtukulusid konverentsil OWLED 2007

Erialane enesetäiendus

- 23.08–05.09.1998 suvekool "Formal Grammars and their Applications", Tartu Ülikool
- 01.–12.03.1999 14th Vilem Mathesius Lecture Series, Charles University, Praha
- 04.–09.03.2001 6th Estonian Winter School in Computer Science, Palmse
- 23.07–03.08.2001 8th Central European Summer School in Generative Grammar, Nish
- 13–24.08.2001 13th European Summer School in Logic, Language and Information, Helsingi
- 18–29.08.2003 15th European Summer School in Logic, Language and Information, Viin
- 01–05.09.2003 Kursus "Language evolution and computation", Simon Kirby and Jim Hurford (University of Edinburgh). Graduate School of Language Technology, Helsingi
- 30.01–01.02.2004 Estonian Computer Science Theory Days, Koke

- 01–05.03.2004 Kursus “Treebanks: Formats, Tools and Usage”, Stockholm University, Stockholm
- 07–20.03.2004 19th Vilem Mathesius Lecture Series, Charles University, Praha
- 09–20.08.2004 16th European Summer School in Logic, Language and Information, Nancy
- 25–29.07.2005 Reasoning Web 2005 Summer School, University of Malta, Msida

List of original publications

[1] James Dowdall, Michael Hess, Neeme Kahusk, Kaarel Kaljurand, Mare Koit, Fabio Rinaldi, and Kadri Vider. Technical Terminology as a Critical Resource. In *Third International Conference on Language Resources and Evaluation (LREC 2002)*, pages 1897–1903, Las Palmas, Canary Islands, Spain, 2002.

[2] James Dowdall, Fabio Rinaldi, Andreas Persidis, Kaarel Kaljurand, Gerold Schneider, and Michael Hess. Terminology expansion and relation identification between genes and pathways. In *Workshop on Terminology, Ontology and Knowledge Representation*, Universite Jean Moulin (Lyon 3), 2004. 7 pages.

[3] Norbert E. Fuchs, Stefan Höfler, Kaarel Kaljurand, Fabio Rinaldi, and Gerold Schneider. Attempto Controlled English: A Knowledge Representation Language Readable by Humans and Machines. In Norbert Eisinger and Jan Małuszyński, editors, *Reasoning Web, First International Summer School 2005, Msida, Malta, July 25–29th, 2005, Revised Lectures*, number 3564 in Lecture Notes in Computer Science, pages 213–250. Springer, 2005.

[4] Norbert E. Fuchs, Kaarel Kaljurand, and Gerold Schneider. Attempto Controlled English Meets the Challenges of Knowledge Representation, Reasoning, Interoperability and User Interfaces. In Geoff Sutcliffe and Randy Goebel, editors, *Nineteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS 2006)*, pages 664–669, Melbourne Beach, Florida, May 11–13th 2006. The AAAI Press, Menlo Park, California.

[5] Neeme Kahusk and Kaarel Kaljurand. *Semyhe* tulemusi: kas tasub *naise* pärast WordNet ümber teha? In Renate Pajusalu and Tiit Hennoste, editors, *Tartu Ülikooli üldkeeleteaduse õppetooli toimetised 3: Tähendusepüüdja / Catcher of the Meaning*, pages 185–195. Tartu University Press, Tartu, 2002.

[6] Neeme Kahusk, Kaarel Kaljurand, Mare Koit, and Kadri Vider. Kasutajaliides info hankimiseks elektroonilisest käsiraamatust: Zürichi ja Tartu ühisprojekt. In Tiit Hennoste, editor, *Tartu Ülikooli üldkeeleteaduse õppetooli toimetised 1: Arvutuslingvistikalt inimesele*, pages 167–182. Tartu University Press, Tartu, 2000.

[7] Kaarel Kaljurand. Word sense disambiguation of Estonian with syntactic dependency relations and WordNet. In Laura Alonso i Alemany and Paul Égré, editors, *ESSLLI-2004 Student Session*, pages 128–137, Nancy, France, 2004.

[8] Kaarel Kaljurand. Piiratud inglise keel ACE ja sellega seotud tarkvara. In Mare Koit, Renate Pajusalu, and Haldur Õim, editors, *Tartu Ülikooli üldkeeleteaduse õppetooli toimetised 6: Keel ja arvuti*, pages 268–278. Tartu University Press, Tartu, 2006.

[9] Kaarel Kaljurand and Norbert Fuchs. Verbalizing OWL in Attempto Controlled English. In Christine Golbreich, Aditya Kalyanpur, and Bijan Parsia, editors, *3rd OWL Experiences and Directions Workshop (OWLED 2007)*, volume 258. CEUR Proceedings, 2007. 10 pages.

[10] Kaarel Kaljurand and Norbert E. Fuchs. Bidirectional mapping between OWL DL and Attempto Controlled English. In José Júlio Alferes, James Bailey, Wolfgang May, and Uta Schwertel, editors, *Fourth Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR 2006)*, number 4187 in Lecture Notes in Computer Science, pages 179–189, Budva, Montenegro, 2006. Springer.

[11] Kaarel Kaljurand and Norbert E. Fuchs. Mapping Attempto Controlled English to OWL DL. In *3rd European Semantic Web Conference. Demo and Poster Session*, pages 11–12, Budva, Montenegro, June 12th 2006.

[12] Kaarel Kaljurand, Fabio Rinaldi, James Dowdall, and Michael Hess. Exploiting Language Resources for Semantic Web Annotations. In *LREC-2004*, pages 815–818, Lisbon, Portugal, 2004.

[13] Fabio Rinaldi, James Dowdall, Michael Hess, Kaarel Kaljurand, and Magnus Karlsson. The Role of Technical Terminology in Question Answering. In *TIA-2003*, Strasbourg, France, 2003. 10 pages.

[14] Fabio Rinaldi, James Dowdall, Michael Hess, Kaarel Kaljurand, Mare Koit, Neeme Kahusk, and Kadri Vider. Terminology as Knowledge in Answer Extraction. In *TKE-2002*, pages 107–112, Nancy, France, 2002.

[15] Fabio Rinaldi, James Dowdall, Michael Hess, Kaarel Kaljurand, Andreas Persidis, Babis Theodoulidis, Bill Black, John McNaught, Haralampos Karanikas, Argyris Vasilakopoulos, Kelly Zervanou, Luc Bernard, Gian Piero Zarri, Hilbert Bruins Slot, Chris van der Touw, Margaret Daniel-King, Nancy Underwood, Agnes Lisowska, Lonneke van der Plas, Veronique Sauron, Myra Spiliopoulou, Marko Brunzel, Jeremy Ellman, Giorgos Orphanos, Thomas Mavroudakis, and Spiros Taraviras. Parmenides: an opportunity for ISO TC37 SC4? In *ACL-2003 workshop on Linguistic Annotation*, Sapporo, 2003.

[16] Fabio Rinaldi, James Dowdall, Michael Hess, Diego Mollá, Rolf Schwitter, and Kaarel Kaljurand. Knowledge-Based Question Answering. In *KES-2003 Knowledge-Based Intelligent Information & Engineering Systems*, Oxford, 2003.

[17] Fabio Rinaldi, James Dowdall, Kaarel Kaljurand, Michael Hess, and Diego Mollá. Exploiting Paraphrases in a Question Answering System. In *ACL-2003 workshop on paraphrasing (IWP2003)*, Sapporo, 2003.

[18] Fabio Rinaldi, Kaarel Kaljurand, James Dowdall, and Michael Hess. Breaking the deadlock. In *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, pages 876–888, Catania, Sicily, Italy, 2003.

[19] Fabio Rinaldi, Thomas Kappeler, Kaarel Kaljurand, Gerold Schneider, Manfred Klenner, Michael Hess, Jean-Marc von Allmen, Martin Romacker, and Therese Vachon. OntoGene in Biocreative II. In Lynette Hirschman, Martin Krallinger, and Alfonso Valencia, editors, *Second BioCreative Challenge Evaluation Workshop*, 2007.

[20] Fabio Rinaldi, Gerold Schneider, Kaarel Kaljurand, James Dowdall, Andreas Persidis, and Ourania Konstanti. Mining relations in the GENIA corpus. In *Second European Workshop on Data Mining and Text Mining for Bioinformatics (in conjunction with ECML/PKDD 2004)*, pages 61–68, Pisa, Italy, September 2004.

[21] Fabio Rinaldi, Gerold Schneider, Kaarel Kaljurand, Michael Hess, Christos Andronis, Ourania Konstandi, and Andreas Persidis. Mining of Functional Relations between Genes and Proteins over Biomedical Scientific Literature using a Deep-Linguistic Approach. *Artificial Intelligence in Medicine*, 39(2):127–136, 2007.

[22] Fabio Rinaldi, Gerold Schneider, Kaarel Kaljurand, Michael Hess, Christos Andronis, Andreas Persidis, and Ourania Konstanti. Relation Mining over a Corpus of Scientific Literature. In *AIME 2005*, number 3581 in Lecture Notes in Artificial Intelligence, pages 550–559, Aberdeen, Scotland, July 2005. Springer.

[23] Fabio Rinaldi, Gerold Schneider, Kaarel Kaljurand, Michael Hess, and Martin Romacker. An Environment for Relation Mining over Richly Annotated Corpora: the case of GENIA. In *SMBM 2006: 2nd International Symposium on Semantic Mining in Biomedicine*, pages 68–75, Jena, Germany, April 9th–12th 2006.

[24] Fabio Rinaldi, Gerold Schneider, Kaarel Kaljurand, Michael Hess, and Martin Romacker. An environment for relation mining over richly annotated corpora: the case of GENIA. *BMC Bioinformatics*, 7(Suppl 3), 2006.

[25] Fabio Rinaldi, Gerold Schneider, Kaarel Kaljurand, Michael Hess, and Martin Romacker. Tools for Text Mining over Biomedical Literature. In *ECAI 2006 poster session*, 2006.

[26] Gerold Schneider, Kaarel Kaljurand, Fabio Rinaldi, and Tobias Kuhn. Pro3Gres Parser in the CoNLL Domain Adaptation Shared Task. In *Workshop on Computational Natural Language Learning (CoNLL-XI) Shared Task*, 2007.

[27] Gerold Schneider, Fabio Rinaldi, Kaarel Kaljurand, and Michael Hess. Steps towards a GENIA Dependency Treebank. In *Treebanks and Linguistic Theories (TLT 2004)*, pages 137–148, Tübingen, Germany, December 2004.

[28] Gerold Schneider, Fabio Rinaldi, Kaarel Kaljurand, and Michael Hess. Closing the Gap: Cognitively Adequate, Fast Broad-Coverage Grammatical Role Parsing. In *2nd International Workshop on Natural Language and Cognitive Science (NLUCS-2005)*, Miami, 24–25 May 2005.

[29] Gerold Schneider, Fabio Rinaldi, Kaarel Kaljurand, and Manfred Klenner. From Probabilistic Dependency Parsing to Biomedical Text Mining. In *31st Annual Conference of the German Classification Society on Data Analysis, Machine Learning, and Applications*, 2007.

[30] Kadri Vider and Kaarel Kaljurand. Automatic WSD: Does it make sense of Estonian? In *SENSEVAL-2: Second International Workshop on Evaluating Word Sense Disambiguation Systems*, pages 159–162, Toulouse, France, 2001.